UNIVERSIDAD CARLOS III DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

INGENIERÍA EN INFORMÁTICA

Proyecto Fin de Carrera

"CREACIÓN DE MMORTS"

Autor: Jose Luis Nieto García

Tutor: Juan Peralta Donate

2012

Título: “Creación de MMORTS”

Autor: Jose Luis Nieto García

Director: Juan Peralta Donate

EL TRIBUNAL

Presidente: _____

Vocal: _____

Secretario: _____

Realizado el acto de defensa y lectura del Proyecto Fin de Carrera el día de de 2012 en Leganés, en la Escuela Politécnica Superior de la Universidad Carlos III de Madrid, acuerda otorgarle la CALIFICACIÓN de

VOCAL

SECRETARIO PRESIDENTE

AGRADECIMIENTOS

A mis padres, cuyo esfuerzo y sacrificio ha permitido que pueda estudiar la carrera que con la entrega de este proyecto llega a su fin. Porque toda mi constancia y afán de crecer intelectualmente por medio de los estudios, nace de cada uno de sus consejos y acciones diarias que he podido tomar como ejemplo. Por estar ahí siempre y por hacerme sentir que su empeño en mi formación, es una de las muchas formas de quererme que siempre han tenido y tendrán.

A mi hermana, mi cuñado y toda de mi familia, por saber que pase lo que pase siempre estarán para y por mí, porque todo lo bueno que pueda tener, procede indudablemente de todos y cada uno de ellos, porque por mucha que sea la distancia o el tiempo que nos separe, saber que siempre podré contar con su apoyo, por compartir conmigo vivencias, experiencias y acontecimientos que han hecho de mi lo que soy hoy, por ayudarme académica y laboralmente con todo lo que estaba en vuestras manos, porque jamás hubiese podido ni querido tener una familia mejor que vosotros y por quererme como nadie lo ha hecho ni hará.

A mi amigo Jose, si no he podido tener la suerte de tener un hermano de sangre, si he tenido la suerte de encontrarlo, por ser una de mis mayores motivaciones en mis estudios, porque su iniciativa por aprender hasta en sus ratos libres nunca dejará de admirarme, por hacerme comprender que todo ese esfuerzo encuentra su recompensa cuando puedes dedicarte a lo que te gusta, por ser un ejemplo a seguir de lo que algún día podría llegar a ser no solo académicamente, sino también como persona.

A mi amigo Sexto, porque sin nuestra infancia pegados día y noche a los monitores de nuestro ordenadores, jugando y riendo, mi motivación por estudiar esta carrera no hubiera sido igual y por compartir una época que a día de hoy sigo añorando.

A mis amigos y compañeros de carrera Oskar y Pako, por las incontables horas trabajando juntos, por las también incontables horas riendo juntos, compartiendo no sólo unos estudios, sino una gran etapa de nuestras vidas, por tirar los unos de los otros hasta superar cualquier desafío o bache que encontrásemos en el camino, ya fuese académico o no, por tener, no una sino dos manos a las que poder agarrarse y continuar, por estar y seguir estando siempre ahí al margen de cualquier carrera o trabajo.

A todos mis amigos Anushka, Petete, Tuner y Andrinal, si uno puede contar los amigos de verdad con los dedos de una mano, yo tengo la suerte de necesitar dos, por estar siempre ahí, en lo malo y en lo bueno, o mejor dicho, en lo mejor. Por empaparnos tanto en alcohol, como en lágrimas, porque gente como vosotros hace que todo sea más fácil y merezca la pena seguir, y porque cualquier cosa que haya podido o pueda alejarnos ha sido y será siempre un error.

A todos esas personas que la Universidad me ha dado la posibilidad de conocer, especialmente a Fidel, Víctor, Elena, Laura, Maya, Irene, Raúl, Alex, Alberto, Luis, Antonio y Javi, por compartir con vosotros mucho más que una carrera, horas de estudio y clases, por formar entre todos vosotros un ambiente que echaré y mucho de menos.

A Sonia y Txabe, por estar junto a mí y de forma desinteresada en algunos tramos de este camino, ayudándome en lo personal y por tanto en lo académico, haciéndome saber que realmente hay personas que valen y mucho, por compartir conmigo momentos que no olvidaré y por ser ese tipo de personas que me hubiese gustado tener siempre cerca.

A mis compañeros de Evalues: Jose María, Miguel y Diego, por motivarme día a día con su actitud y forma de trabajar, por ayudarme a acabar mis estudios, por formar entre todos un ambiente de trabajo

que, sea cual sea mi futuro laboral, seguro añoraré y envidiaré y por demostrarme que ser grandes profesionales y personas puede ir de la mano.

A mi tutor, por su gran paciencia, comprensión y ayuda durante el tiempo que ha durado este proyecto, por darme plena libertad, guiarme y motivarme fuese cual fuesen mis ideas e iniciativas.

A la familia de Almu, por motivarme, incentivarme y apoyarme en acabar mi carrera, porque hubo un tiempo en el que tuve la suerte de despertarme por las mañanas y sentir que tenía dos familias, por hacerme sentir durante ese tiempo que aunque cayese, debajo siempre habría una red y porque aunque las cosas no salen como uno espera o desea, nunca caen en el olvido.

A todos aquellos que me deje en el tintero, pero que alguna vez y de alguna forma, me ayudaron a llegar hasta donde he llegado.

Gracias.

ÍNDICE

1. INTRODUCCIÓN

1.1. DESCRIPCIÓN DE LOS CAPÍTULOS DEL PROYECTO

Este documento tratará la problemática encontrada durante el diseño e implementación de un videojuego masivo online de estrategia en tiempo real (**MMORTS**). La estructura del mismo es la siguiente:

- **Introducción:** Presentación del problema y recopilación de las definiciones de los términos utilizados en el documento.
- **Objetivos de la cuestión:** Definición de los objetivos perseguidos en el proyecto.
- **Estado de la cuestión:** Estudio del actual mundo del videojuego y de su apartado multijugador en red; introducción a los conceptos necesarios de redes de ordenadores, de persistencia de datos, de modelado tridimensional y de las librerías y motores de videojuegos. Se analizarán las alternativas planteadas para realizar a posteriori la elección más adecuada.
- **Análisis:** Descripción de la funcionalidad de la aplicación una vez escogida la tecnología a utilizar. Se recoge con detalle mediante una lista de requisitos de usuario, casos de uso y requisitos software. Se incluye también un diagrama de estados que enumera los distintos estados de funcionalidad de la aplicación y las transiciones entre ellos.
- **Diseño:** Detalla cómo va a ser implementada la funcionalidad requerida para la aplicación, valiéndose para ello de diagramas de subsistemas y componentes, de despliegue, de clases y bases de datos, convenientemente comentados.
- **Implementación:** Desde un nivel de abstracción más bajo, se explica cómo se han abordado los aspectos implicados en el desarrollo de cada una de las partes destacables de la aplicación.
- **Conclusiones y futuros desarrollos:** Recoge las conclusiones obtenidas como parte de la culminación del proyecto, además de una lista de posibles ideas a desarrollar para ampliarlo en un futuro.
- **Planificación y presupuesto:** Detalla la distribución de recursos en el periodo de tiempo dedicado al proyecto y el coste asociado al mismo.
- **Glosario:** Listado de términos y acrónimos junto a sus definiciones, que han sido usados a lo largo del documento.
- **Referencias:** Listado de fuentes que han sido utilizadas para el desarrollo del proyecto y de este documento.

2. OBJETIVOS DEL PROYECTO

El primordial objetivo del proyecto de fin de carrera aquí tratado, es el estudio de los problemas que conlleva el desarrollo de un videojuego multijugador masivo y los asociados a la infraestructura requerida para dar soporte a una gran cantidad no prestablecida de usuarios y variable a lo largo del tiempo.

La afición al mundo de los videojuegos, la inquietud por aprender cómo desarrollar uno, y por participar en su implementación, y el interés por el ámbito de las redes de ordenadores, conforman la motivación personal seguida para escoger la temática del proyecto.

Partiendo de estas dos premisas, se ha desarrollado para este proyecto de fin de carrera un manual en el que se presentan:

- Conceptos teóricos sobre los que se sostiene la solución escogida en relación a la infraestructura que da soporte a un videojuego multijugador masivo.
- Análisis de los imprevistos vinculados a este género de videojuegos.
- Estudio de las posibles soluciones para su implementación.

Por último, el manual presentará un ejemplo de implementación realizada con las soluciones más viables y óptimas de entre todas las expuestas.

3. ESTADO DE LA CUESTIÓN

Este apartado describirá los conceptos teóricos necesarios para abordar el diseño e implementación de un video juego de tipo MMORTS. Esto supone tratar conceptos acerca de los juegos multijugador y de estrategia, de comunicaciones, de persistencia de datos, de modelado tridimensional y de librerías y motores destinados al desarrollo de video juegos.

3.1. RESUMEN DEL ESTADO ACTUAL DEL ENTRETENIMIENTO DIGITAL

En la industria del entretenimiento, el sector de los videojuegos es a día de hoy uno de los más rentables, obteniendo incluso mayores beneficios que el cine. Los videojuegos han logrado una revolución en el sector del entretenimiento que además, no se detiene en su incesante evolución y crecimiento hacia nuevas áreas, como lo son las comunicaciones a través de internet.

Para encontrar el origen de la industria del videojuego debemos remontarnos hasta 1972, cuando apareció la primera videoconsola llamada **Magnavox Odyssey** (a manos de la empresa Magnavox), sistema que permitía a sus usuarios jugar a un deporte similar al tenis.

No obstante, sería ese mismo año cuando Nolan Bushnell y Ted Dabney fundaron **Atari** [1], que siendo la primera empresa orientada a los videojuegos ocasionaría el origen de la industria del videojuego. La empresa basó su crecimiento inicial en el desarrollo de máquinas recreativas de monedas, que permitían a sus usuarios jugar al juego **Pong**. El éxito fue tal que la compañía decidió lanzar una consola doméstica en 1975 que permitiese jugar al popular juego y cuyo nombre sería también Pong.

Ilustración 1: Máquina recreativa de Atari del juego Pong

A finales de los años 70, ambas compañías (Magnavox y Atari) depositaron sus esfuerzos en la creación de un único sistema que permitiese jugar a diversos juegos. Los frutos de dicho esfuerzo fueron la Odyssey 2 y la Atari 2600. Aunque el verdadero éxito de Atari llegaría más tarde con la salida al mercado de **Space Invaders** y su culmen con la salida de Pac-Man.

Ante el palpable éxito obtenido en el mercado por Atari, surgieron muchas compañías que pretendían competir para obtener sus propios beneficios en el sector. Comenzó de esta forma una guerra entre Atari, Mattel, **Nintendo** [2] y **Sega** [3] que trajo consigo la creación de consolas como la Mattel Intellivision 1980, la Atari 5200, la Sega Sg-1000 (que posteriormente acabaría siendo la **Sega Master System**) y la **Nintendo Entertaiment System**.

La llegada al mercado de cada una de ellas suponía una evolución en el hardware visto hasta el momento, ya que este era la principal característica que las compañías podían explotar para competir con el resto. Cabe destacar el éxito logrado por la NES de 8 bits respecto al resto de consolas, debido principalmente al juego **Mario Bros**.

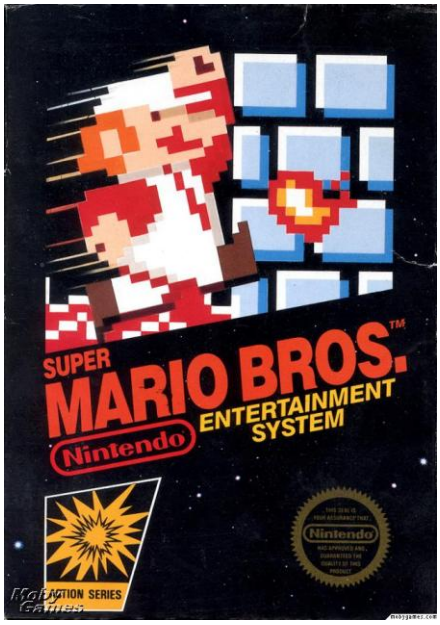


Ilustración 2: Primer Mario Bros lanzado por Nintendo

Sega lanzó en 1989 la **Mega Drive**, la primera consola de 16 bits que doblaba en potencia al resto de consolas competidoras. La respuesta de Nintendo llegó tras tres años con **Super Nintendo**, que fue incapaz de frenar el dominio de Sega hasta que Squaresoft y Nintendo aunaran sus fuerzas y sacaran al mercado el juego **Donkey Kong Country**.

Atari que por aquel entonces se encontraba eclipsada por la competencia de Nintendo y Sega, intentó un movimiento maestro comprando **Handy** a la compañía **Epyx**, la primera videoconsola portátil creada. Distribuida bajo el nombre de **Lynx** y con unas características tecnológicas muy superiores a las vistas hasta la fecha, no logró el resultado esperado en el mercado. Las causas fueron principalmente sus deficiencias respecto a consumo de batería, tamaño y precio. La respuesta de Sega y Nintendo no tardaría en llegar en forma de **GameGear** y **GameBoy**.

Ilustración 3: Primera consola portátil Lynx

La guerra acabó para Atari y Mattel, que se resignaron a desarrollar únicamente juegos. Fue en 1995 cuando **Sony** llegara a la lucha que mantenían Sega y Nintendo, provocando una revitalización del sector caracterizada por las consolas de 32 bits. La **Saturn** de Sega fue un desastre en ventas respecto a la **Play Station** de Sony, que consiguió apartar a la compañía japonesa del mercado. Por su parte, la espera de un año de Nintendo para lanzar su sistema al mercado, trajo consigo la primera videoconsola de 64 bits. Aun siendo claramente superior a sus competidoras, los cartuchos de los que hacía uso la **Nintendo 64** frente a los CD-ROM de Sony, ocasionaron un escaso éxito en ventas.

El último intento de Sega para recuperar su trono en el mercado de consolas fue con **Dreamcast** en el año 1999, un prematuro lanzamiento que además disponía de una potencia nunca vista. Resultó un fracaso y fue la última consola lanzada por Sega, debido principalmente a los rumores sobre el nuevo sistema de Sony y al lastre arrastrado por Saturn. Dos años más tarde llegaron los sistemas de Sony (**PlayStation 2**), de Nintendo (**GameCube**) y de la naciente Microsoft (**Xbox**).

Ilustración 4: Dreamcast, la última consola de Sega

En la actualidad, ante la inminente llegada de la nueva generación de consolas y tras 40 años de evolución, la **Xbox 360** de Microsoft, la **Wii** de Nintendo y la **PlayStation 3** de Sony compiten en una lucha muy igualada en la que Microsoft parece haber logrado una pequeña ventaja sobre el resto.

3.2. EVOLUCIÓN DEL MULTIJUGADOR

La evolución del multijugador en los videojuegos ha resultado más que notoria desde su nacimiento en 1969, año en el que el primer juego online vio la luz a manos de Rick Blomme. El juego daba la posibilidad a sus jugadores de compartir la experiencia de juego de **Spacewar**, desarrollado por estudiantes del **MIT** sobre el sistema *PLATO*.

El juego disponía de un sistema **BBS** [4] que por aquella época y debido a las deficiencias de la tecnología, no conseguía mantener actualizadas los movimientos simultáneos de los jugadores, lo que ocasionaba que los usuarios interactuaran sobre estados del juego que podrían estar obsoletos en ese momento.

A raíz de este tipo de sistema surgieron los primeros juegos de aventuras en texto, lo conocidos como **MUD** y que estaban basados en los juegos RPG de papel y lápiz. El papel de director de este tipo de juegos era llevado a cabo por el administrador del sistema BBS, encargado de controlar los turnos de los jugadores.

El primer videojuego MUD vio la luz en 1979,que alojado en la universidad de Essex, permitía a los jugadores interaccionar mediante mensajes de texto con el mundo, con personajes dotados de inteligencia artificial y con otros jugadores que se encontrasen jugando.

La siguiente evolución llegó con la inserción de un apartado multijugador en los juegos de acción en primera persona (**FPS**). El primer juego que puede ser catalogado dentro de esta categoría fue **A-maze-ing** para Macintosh en 1989, aunque sería **Doom** de **ID Software** el mayor precursor y buque insignia de esta categoría. Los gráficos tridimensionales de Doom fueron algo nunca visto hasta la fecha y su éxito fue tal que supuso el comienzo de una saga que continuó con **Doom II** (Actualmente Doom 4 se encuentra en desarrollo). El juego de ID Software, por aquel entonces ya soportaba partidas multijugador de hasta 4 jugadores.

Propiciado también por el éxito de Doom, ID Software lanzó **Quake** al mercado, que ampliaba notoriamente la experiencia multijugador dando soporte hasta a 16 jugadores simultáneos.

Ilustración 5: Videojuego Doom de ID Software

De una forma paralela a los FPS, comienzan a brotar los primeros juegos de estrategia por turnos y los juegos basados en navegadores Web como **Terranlegacy**. La categoría evolucionó dando el paso más lógico, la estrategia por turnos pasó a ser estrategia en tiempo real. **Blizzard** fue la que tomó la iniciativa lanzando **Warcraft: Orcs & Humans** que ofrecía a sus jugadores la posibilidad de compartir partidas entre dos usuarios a través de internet.

Aunque el verdadero nacimiento del género multijugador masivo se produjo en el año 1997 con la salida al mercado de **Ultima Online**, que a pesar de ser posterior a **Air Warrior**, supuso el punto de inflexión esperado para las características online que ofrecía.

Ultima Online se puede enmarcar dentro del género de rol multijugador masivo en línea. El juego se desarrolla en un mundo virtual de fantasía en el que los jugadores pueden desarrollar sus habilidades y estadísticas, incluye además un sistema económico basado en la oferta y demanda de la que los jugadores hacen uso.

Siguiendo la estela de este último, **Everquest** fue desarrollado y puesto a la venta por la compañía **Verant Interactive**. Aunque en una atmósfera más futurista, el juego ofrecía características que distaban poco de las ofrecidas por Ultima Online.

Los últimos pasos evolutivos de los juegos multijugador masivos llegaron en 2003, manteniendo sus principales características hasta el día de hoy. **Second Life** a manos de **Linden Labs**, intentaba recrear la experiencia de la vida real en un mundo virtual futurista. La interacción entre jugadores, la creación de negocios, la construcción de casas y la adquisición de bienes fueron algunas de las posibilidades más innovadoras ofrecidas por el juego.

3.3. COMUNICACIONES

3.3.1. REDES DE ORDENADORES

Se define como red de ordenadores al conjunto de dos o más ordenadores interconectados por medio de cualquier medio de transmisión de datos (cable, señales, etc.), lo que les permite la compartición de información y/o programas.

La clasificación de las redes de ordenadores se realiza en torno a cinco criterios:

- **Alcance**: **PAN**, **LAN**, **MAN**, **WAN**.
- **Método de conexión**: Cable coaxial, par trenzado, fibra óptica, radio, infrarrojos o laser.
- **Relación funcional**: Cliente-Servidor, Par a Par.
- **Topología de red**: Red en bus, estrella, anillo, malla, árbol o mixta.
- **Direccionalidad de los datos**: Simplex, Half-duplex y Full-Duplex [5].

La conexión entre computadores se realiza por medio de tarjetas de red que permitan el envío y recepción de paquetes. Dichos paquetes deben respetar un protocolo que definirá su formato para poder ser transmitidos. Las tarjetas de red que interconectan los equipos, disponen de un identificador único de 48 bits denominado **MAC** con el que se direccionan los paquetes entre los diversos computadores que conforman la red.

3.3.2. SISTEMAS DISTRIBUIDOS

Se define con este nombre al conjunto de computadoras conectadas entre sí, las cuales disponen de sus propios componentes hardware y software, que son percibidos por el usuario como un único sistema. La fiabilidad de los sistemas distribuidos es muy elevada, ya que el fallo en el funcionamiento de un componente de cualquiera de los equipos, puede ser remplazado por el funcionamiento de otro componente en otro equipo.

Las principales característica de un sistema distribuido son las siguientes:

- **Compartición de recursos**: Los recursos de cada equipo, ya sean de tipo físico (hardware) o de tipo lógico (programas) se encuentran encapsulados físicamente en cada una de las computadoras. El acceso a estos recursos por parte de otras computadoras se realiza por medio de protocolos de comunicación.
- **Concurrencia**: Consiste en la ejecución de varios procesos en una misma máquina, ya sea porque varios clientes quieran acceder a un mismo recurso, o porque se ejecutan en el mismo equipo varias aplicaciones. Este término se explica en detalle en el apartado **3.1.2**.
- **Escalabilidad**: El número de equipos del sistema distribuido no debe influir en el software del mismo, por lo que un aumento o disminución de equipos no debe influir en la eficiencia y efectividad, lo que ocasiona problemas relacionados con las prestaciones de la red, del hardware y con los aspectos del diseño orientado a garantizar la escalabilidad.
- **Tolerancia a fallos**: Mediante redundancia hardware (uso de componentes redundantes) y recuperación del software (programación de software capaz de recuperarse frente a fallos) se consigue que los sistemas sean capaces de evitar el colapso y puedan recuperarse con relativa facilidad.
- **Transparencia**: Ocultación al usuario de los componentes hardware del sistema distribuido, se percibe como un todo en lugar de como un conjunto de componentes independientes.

3.3.3. MODELO OSI

La Organización Internacional para la Estandarización [6] creo el modelo de referencia **OSI** como un marco de referencia para definir arquitecturas de sistemas de comunicación y solventar así, los problemas presentados en la comunicación por las diferencias de los diversos equipos. No puede considerarse un estándar de comunicación, pero ha sido utilizado como base para la creación de estándares y protocolos.

El modelo OSI define una estructura de varios niveles, de forma que cada nivel o capa resuelve una parte del problema de comunicación. El modelo se encuentra dividido en las siguientes capas:

Ilustración 6: Modelo OSI

A continuación se detalla cada uno de estos niveles:

- **Capa física**: Destinada a transformar las tramas de datos recibidas por la capa de enlace y transmitirlas a nivel de bit, controlando las propiedades físicas y eléctricas de los componentes hardware.
- **Capa de enlace**: Se encarga del direccionamiento físico, de la topología de la red, del acceso a la misma, de la notificación de errores, de la distribución ordenada de tramas y del control de flujo.
- **Capa de red**: Este nivel realiza el direccionamiento lógico y determina la ruta de los datos hasta alcanzar su destino, incluso cuando los nodos no se encuentren conectados de forma directa. Incluye por tanto enrutamiento y envío de paquetes entre redes.
- **Capa de transporte**: Destinada a transportar datos de la máquina origen a la de destino, sin tener en cuenta el tipo de red que se utilice.
- **Capa de sesión**: Mantiene el enlace entre dos equipos que se encuentran transmitiendo datos de forma simultánea.
- **Capa de presentación**: Encargada de manejar estructuras de datos abstractas y realizar las conversiones necesarias para que puedan ser debidamente interpretadas por los distintos equipos.
- **Capa de aplicación**: Define los protocolos utilizados por las aplicaciones para intercambiar datos.

3.3.4. PROTOCOLOS DE NIVEL DE TRANSPORTE

Protocolo UDP

UDP es un protocolo que permite el envío de datagramas a través de la red sin que haya prestablecida una conexión entre los equipos. Únicamente es necesario que la aplicación que envía información disponga de la dirección IP y el puerto al que se dirige, y que la aplicación que recibe esté escuchando en el puerto conocido.

- **Ventajas**:
  - El datagrama contiene todos los datos necesarios para llegar a su destino. No es necesaria una conexión previa.
  - No existen retardos en el envío, ya que no ofrece garantías en la recepción.
- **Desventajas**:
  - No existen mecanismos de confirmación de recepción, por lo que no se garantiza la llegada del mensaje al destino.
  - No consta de control de flujo, por lo que el orden de los mensajes puede ser distinto al de envío.

A fines prácticos, el protocolo se utiliza cuando la velocidad tiene mayor importancia que asegurar la llegada de los mensajes, ya que cualquier mecanismo de control introduce retardos en la comunicación. Un claro ejemplo sería el **Streaming** de audio y video.

Protocolo TCP

Protocolo que permite el envío de mensajes a través de la red, habiendo establecido una conexión previamente.

- **Ventajas:**
  - Se garantiza la llegada de los paquetes al otro extremo (Fiabilidad).
  - Los paquetes llegan a su destino en el orden de envío (Control de Flujo).
- **Inconvenientes:**
  - El establecimiento de la conexión previa a la comunicación y el cierre de la misma al finalizar.

El ciclo de vida natural de una conexión TCP se puede describir en tres pasos fundamentales:

- **Inicio de la Conexión**: Se establece la conexión entre los dos nodos mediante el Three-Way Handsake [7].
- **Envío de información**: La transmisión de datos se realiza siempre con mensajes de control para asegurar la correcta recepción de los datos.
- **Cierre de la conexión**: Una vez la conexión se dé por finalizada, es necesario indicarlo en ambos extremos de forma implícita.

En vista de lo expuesto, este protocolo se utilizará cuando se conceda mayor importancia a asegurar la recepción de los datos, que a la velocidad con la que se transmiten. La navegación Web es claro ejemplo del uso de este protocolo.

3.3.5. MODELO TCP/IP

DARPA, responsable del desarrollo del modelo TCP/IP, se basó en el modelo OSI para definir cada uno de las capas de TCP/IP. La correspondencia entre los modelos OSI y TCP/IP no es directa, ya que como puede observarse en la ilustración, existen capas de OSI que son implementadas por una o varias capas TCP/IP.

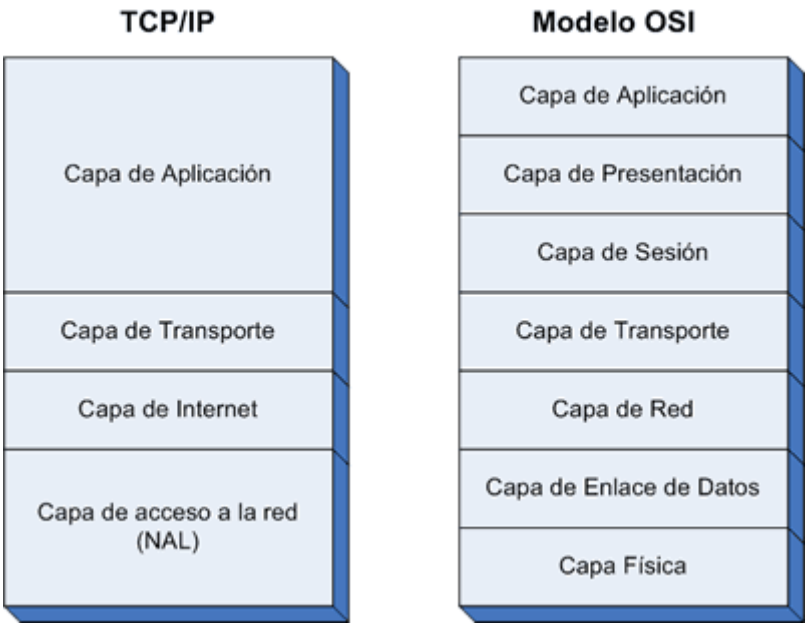


Ilustración 7: Correspondencia entre capas TCP/IP y OSI

Cada de red

Implementa la forma de enviar mensajes de forma física a través de la red, atendiendo a fundamentos de señales eléctricas y a su discretización en bits mediante componentes hardware.

Capa de internet

Implementa el empaquetado de los datagramas IP con información de las direcciones de origen y destino, mediante las cuales se direccionan los datagramas entre redes.

Capa de transporte

Aspectos como sesiones de comunicación entre equipos y el estado de la conexión son implementados en esta capa.

Capa de aplicación

Define la forma en la que los programas de host se conectan con los servicios de nivel de transporte.

El establecimiento de la comunicación por medio de del protocolo TCP/IP, requiere la dirección IP del equipo donde se encuentra la aplicación con la que se quiere conectar, el puerto donde la aplicación escucha y el tipo de protocolo que se quiere utilizar. Cada dirección IP identifica de forma unívoca a cada equipo de la red, por lo que es vital disponer de dicha dirección para iniciar cualquier tipo de comunicación.

Para el caso de IPv4 la dirección la forman cuatro bytes, lo que se traduce en cuatro números comprendidos en el rango [0, 255] y separados por un punto. 192.168.0.1 o 163.117.205.22 son ejemplos válidos de dirección IP.

Los puertos utilizados por los protocolos de capa de aplicación, están orientados a identificar aplicaciones y están compuestos por dos bytes, lo que se traduce en números comprendidos en el rango [0, 65535]. Los puertos de TCP y UDP son independientes, por lo que un mismo puerto puede utilizarse con ambos protocolos de forma simultánea.

3.3.6. PROTOCOLOS DE NIVEL DE APLICACIÓN

Son los destinados al intercambio de datos entre aplicaciones, existen multitud de protocolos definidos para un cometido o tipo de tráfico característico. De esta forma podemos encontrarnos protocolos tales como:

- **FTP:** Protocolo de transferencia de archivos (*File Transfer Protocol*). Existe una versión más ligera del protocolo denominada **TFTP**.
- **DNS:** Servicio de nombres de dominio (*Domain Name Service*). Utilizado para traducir los nombre de dominio en direcciones IP.
- **HTTP:** Protocolo para acceder a páginas web (*Hypertext Transfer Protocol*). Uno de los más famosos y utilizados debido al auge de la web.
- **SMTP:** Protocolo de correo electrónico (*Simple Mail Transport Protocol*). Junto con **POP** conforman los más utilizados protocolos de correo electrónico.
- **SSH:** Interprete de ordenes seguras (*Secure Shell*). Su uso está orientado a la administración de equipos de forma remota. Su propósito es similar a **Telnet** aunque esté último está menos extendido.
- **LDAP:** Protocolo ligero de acceso a directorios (*Lightweight Directory Access Protocol*).

Muchos de estos protocolos disponen a su vez de versiones seguras, es decir, donde el intercambio de información se realiza utilizando mecanismos de seguridad informática (**HTTPS**, **SFTP**, etc.).

En ocasiones el diseñador puede optar por utilizar un protocolo de nivel de aplicación y adaptarlo a las necesidades de su software, ya que de esta forma, se solucionan problemas que podrían surgir en capas más bajas y se ahorra el diseño de un protocolo de aplicación específico.

3.3.6.1. HTTP

Desarrollado por el World Wide Web Consortium [8] y la Internet Engineering Task Force [9], se trata de un protocolo de nivel de aplicación que funciona bajo TCP, que está orientado a transacciones y que sigue el esquema petición-respuesta entre un cliente y un servidor. La respuesta puede tener cualquier tipo de formato (archivos, el resultado de la ejecución de un programa en el servidor, etc.), por lo que se definen tipos **MIME** para que puedan ser interpretados debidamente por el receptor.

Las principales características del protocolo son:

- La comunicación entre cliente y servidor se realiza por medio de caracteres de 8 bits, por lo que es posible transmitir cualquier tipo de documento (texto, binario, etc.).
- Es posible transmitir formatos multimedia siempre que se encuentren clasificados por un tipo MIME.
- Aunque existen muchos verbos o comandos, en general se utilizan tres de ellos en el diálogo entre cliente y servidor:
    - **GET:** El cliente solicita un determinado recurso.
    - **POST:** El cliente envía información al servidor.
    - **HEAD:** El cliente solicita las características o atributos de un recurso.
- Cada operación de HTTP requiere una conexión TCP que es liberada al finalizar la misma. Esta deficiencia ha mejorado hasta permitir mantener las conexiones abiertas durante un periodo de tiempo [10].
- No se mantiene el estado de la comunicación, cada operación no influye en las anteriores ni en las sucesivas. En caso de ser necesaria esta funcionalidad, deberá ser implementada por el programador o ser almacenada en el cliente mediante el uso de **Cookies**.
- Los recursos sobre los que se aplican los comandos, están identificados por medio de la información situada al final de la **URL**.

El ciclo de vida de una petición HTTP puede resumirse en los siguientes pasos:

1. El usuario solicita un determinado recurso accediendo a la URL del mismo. Con la información contenida en dicha URL se resuelve el protocolo DNS para obtener la dirección IP del servidor (el puerto por defecto es el 80).
2. Se realiza la conexión TCP/IP con el servidor y se construye la petición enviando el comando GET o POST, la ubicación del recurso requerido, la versión del protocolo HTTP utilizada e información adicional (datos opcionales para el servidor, características del cliente, etc.).
3. El servidor responde al cliente mediante un código que representa el resultado de la operación, un tipo MIME y el contenido del recurso (siempre que el resultado sea satisfactorio).
4. Se cierra la conexión TCP a excepción de que se utilice HTTP Keep Alive.

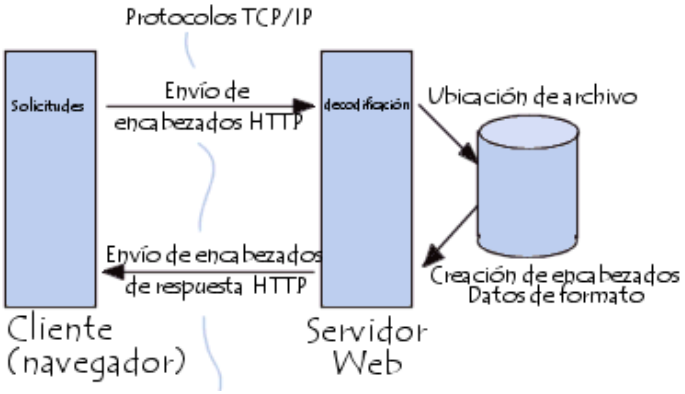


Ilustración 8: Petición y respuesta HTTP

El diálogo entre cliente y servidor se realiza por medio de mensajes formados por líneas de texto plano en las que aparecen los diferentes comandos y opciones del protocolo. Existen únicamente dos tipos de mensajes: Mensajes de petición y mensajes de respuesta, cuya estructura es la siguiente:

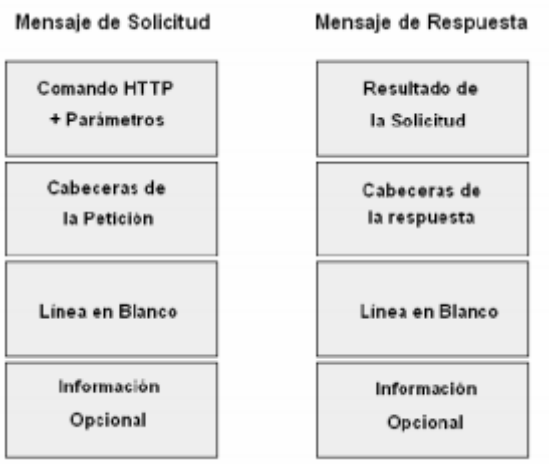


Ilustración 9: Tipos de mensajes HTTP

La primera línea del mensaje de solicitud contiene el verbo o comando que el cliente quiere realizar, mientras que en el mensaje de respuesta nos encontramos con el resultado de la operación (éxito o fracaso). Las cabeceras que encontramos a continuación (con atributos opcionales y obligatorios) condicionan el funcionamiento del protocolo. Por último, la información de los recursos comunicados se ubica al final de los mensajes y separados por el carácter retorno de carro (CR-LF).

3.1.1. TIPOS DE ARQUITECTURAS DE RED

La arquitectura de red definirá los cimientos del diseño de un sistema de comunicación y el plan seguido para conectar los distintos protocolos utilizados. Las arquitecturas de red más utilizadas son:

Arquitectura cliente-servidor

En este tipo de arquitectura aparecen dos tipos de roles que se asignarán a los diversos equipos que la conformen. Cada uno de estos roles es:

- **Cliente:** Realiza las peticiones y espera la respuesta del servidor. Tiene un papel activo en la comunicación.

- **Servidor:** Espera a que le lleguen peticiones de los clientes para posteriormente procesarlas y enviar las respuestas. Su papel en la comunicación es pasivo y acepta peticiones de un elevado número de clientes.

La implementación del servidor puede ser o no concurrente. Una implementación concurrente dará servicio a varios clientes al mismo tiempo mediante el uso de procesos ligeros, mientras que una no concurrente atenderá a sus clientes de uno en uno según el orden de llegada.

Las principales características de este tipo de arquitectura son las siguientes:

- **Escalabilidad:** El número de clientes y servidores son independientes en este tipo de arquitectura.
- **Centralización:** El estado del sistema depende únicamente del servidor, por lo que su integridad no puede verse afectada por los clientes.
- **Mantenimiento:** El mantenimiento es bajo debido a que las funciones del servidor son fácilmente distribuibles en varios equipos y el remplazo ante errores se realiza con relativa sencillez.
- **Robustez:** La falta de disponibilidad del servidor, ocasionada por elevado número de peticiones, puede hacer que se pierdan algunas de estas.
- **Hardware:** El servidor debe ser capaz de soportar la carga de un gran número de clientes, por lo que los costes del hardware asociado a dicho equipo suelen ser elevados.

Arquitectura par a par

Puede considerarse un caso específico de la arquitectura cliente servidor, en la que todos los equipos asumen el rol de cliente y servidor. Este tipo de redes busca optimizar el ancho de banda de los nodos, por medio de establecer conexiones entre ellos de acuerdo a algoritmos de optimización y balanceo de carga.

Las características de este tipo de arquitectura son las siguientes:

- **Escalabilidad:** Este tipo de arquitectura es altamente escalable y además, un elevado número de usuarios favorece a los recursos globales del sistema.
- **Balanceo de carga:** La repartición de la carga es homogénea y equitativa entre los usuarios del sistema.
- **Hardware:** Todos los nodos de la red se consideran iguales respecto a funcionalidad y por tanto, no es necesario ningún tipo de hardware de altas prestaciones en ninguno de los equipos.
- **Robustez:** El fallo de uno de los nodos de la red no supone el colapso de la misma, ya que son fácilmente remplazable por otros.
- **Seguridad:** Son necesarios mecanismos de seguridad, debido a que la información se encuentra replicada en muchos nodos de los que no es posible asegurar sus buenas intenciones.
- **Localización:** Se requieren equipos destinados a proporcionar la localización e información de los nodos que conforman la red.

Atendiendo a la distribución de sus nodos, pueden distinguirse los distintos tipos de redes en este tipo de arquitectura:

- **Red centralizada:** Este tipo de distribución consta de un nodo central que sirve de enlace entre el resto de nodos, almacena los contenidos y los distribuye. Favorece la permanencia del contenido, aunque limita la privacidad de los usuarios y la escalabilidad del sistema.
- **Red descentralizada:** De forma similar a la anterior, la red dispone de uno o más servidores centrales encargados de administrar la comunicación entre nodos, aunque sin compartir

contenidos (no almacena información alguna). Si los nodos centrales cayeran, el resto de nodos disponen de una conexión directa entre ellos.

- **Red distribuida:** Son las redes más comunes, ya que no requieren de nodos centrales. Por ello, descentralizan la administración y almacenamiento a lo largo de los nodos de la red que además, se encuentra conectados entre ellos.

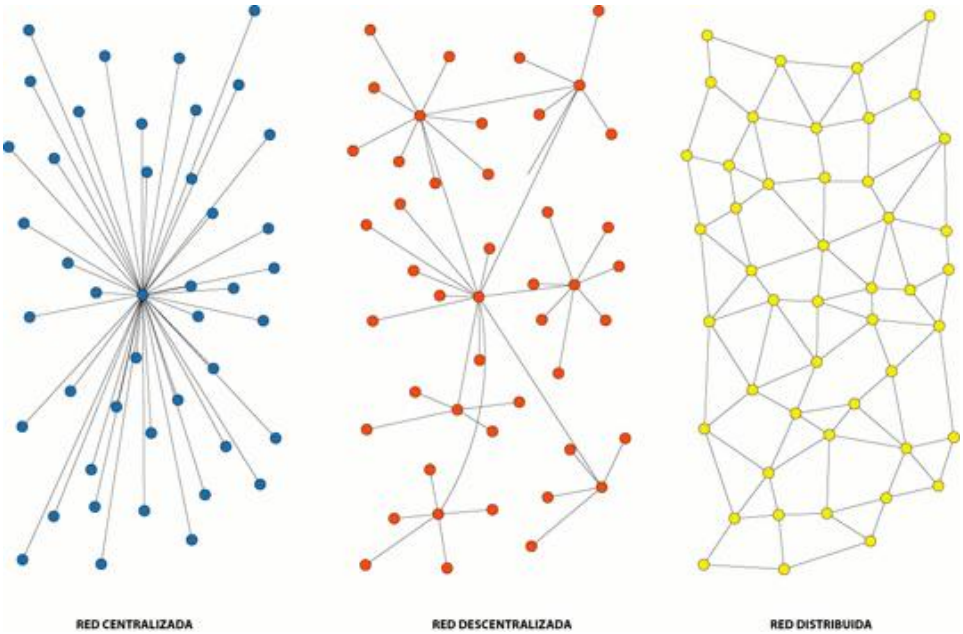


Ilustración 10: Tipos de redes atendiendo a sus nodos

Arquitectura N-Tiered

La arquitectura descrita como Cliente Servidor, puede considerarse del tipo "two-tier" si separamos en dos capas la lógica de la aplicación. Las arquitecturas N-Tiered realizan subdivisiones de la lógica de la aplicación en todas las capas que sean necesarias (separación por ejemplo de los datos y de la interfaz).

Arquitectura Clúster

Está formada por un grupo de ordenadores que se encuentran conectados y que en apariencia y a efectos lógicos, funcionan como un único ordenador. En términos generales, este tipo de arquitectura aumenta la rapidez y la disponibilidad respecto a una única máquina. Sus principales características son las siguientes:

- **Disponibilidad:** Ofrecen sistemas con una alta tolerancia a fallos.
- **Rendimiento y eficiencia:** Ofrecen grandes prestaciones computacionales y una gran cantidad de memoria.
- **Escalabilidad:** Sencillez a la hora de incrementar el número de equipos que forman el clúster.

Los elementos que participan en una arquitectura tipo clúster son los siguientes:

- **Nodos:** Equipos que conforman la red y que disponen de características similares. De esta forma es posible realizar un buen balanceo de carga.
- **Conexiones de red:** Elementos que comunican los distintos equipos del clúster. Habitualmente se hace uso de adaptadores Ethernet, aunque debido a la latencia puede requerirse el uso de redes de mayor velocidad.

- **Middleware:** Software ubicado entre los sistemas operativos y las aplicaciones. Es el encargado de establecer y mantener las conexiones entre los equipos, de balancear la carga de trabajo entre los mismos y de generar la sensación de que la red funciona como un único equipo, atendiendo siempre a términos de eficiencia y rendimiento.
- **Sistemas operativos:** Multiprocesador y multiusuario, deberán ejecutar el middleware.
- **Aplicaciones:** Son el software que se quiere ejecutar para llevar a cabo el procesado del trabajo.

El uso de este tipo de arquitectura ha evolucionado con el tiempo, su uso abarca hoy en día desde aplicaciones orientadas a supercomputación, a servidores Web, bases de datos de alto rendimiento, etc.

3.1.2. CONCURRENCIA

La concurrencia adquiere dos significados distintos, por una parte hace referencia a la capacidad de los ordenadores de ejecutar aplicaciones de forma paralela o de forma simultánea. La segunda definición se vincula a la capacidad de interacción y comunicación, para acceder a un recurso que es compartido, entre procesos que son ejecutados en el mismo periodo de tiempo.

Podemos encontrar concurrencia a nivel de proceso y a nivel de hilo:

- Los **procesos** son programas en ejecución que constan de instrucciones a ejecutar, de un estado que se almacena en los registros del equipo donde se ejecuta, de memoria reservada para el uso que requiera su ejecución y de información propia que es utilizada para llevar a cabo su planificación.
- Los **hilos** por su parte, son ramificaciones de un mismo proceso que se ejecutan de forma paralela. Su principal diferencia respecto a los procesos es que comparten memoria y recursos (archivos abiertos, situación de autenticación, etc.).

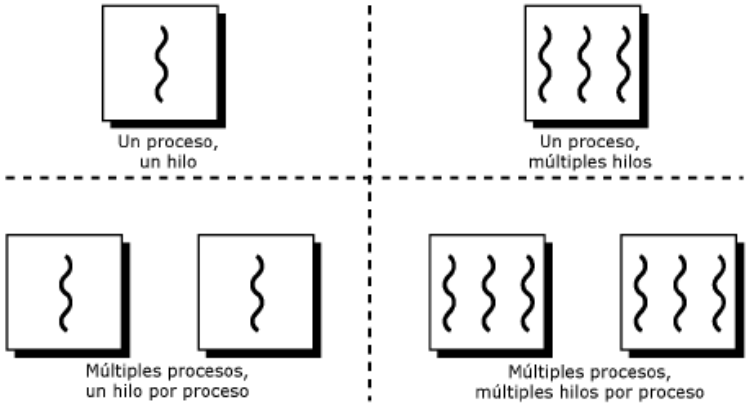


Ilustración 11: Procesos e Hilos

La principal ventaja de diseñar una aplicación con técnicas de ejecución en paralelo, es la de evitar los tiempos de espera ocasionados por las colas de peticiones que recibe una aplicación. Aunque en términos generales la aplicación utilice el mismo tiempo para ejecutar sus tareas de forma paralela o no, la transición entre dichas tareas es simultánea.

Centrándonos en los intereses de este proyecto y aproximándonos a la arquitectura cliente servidor, una aplicación que sea ejecutada en un servidor de forma paralela, provoca que los clientes puedan recibir la respuesta a sus peticiones sin esperar la cola generada por su orden de llegada.

El principal inconveniente del diseño de aplicaciones paralelas, se centra en los problemas que surgen en el diseño de la aplicación y al tiempo que se pierde en la comunicación y sincronización de los procesos. Esto se debe en casi su totalidad a la compartición de datos que realizan los procesos ligeros, ya que garantizar la exclusividad de acceso y mantener los datos en un estado consistente, requiere del uso de mecanismos de control y de un diseño específico para cada caso. Los sistemas operativos actuales ofrecen toda clase de mecanismos para solventar esta problemática (mutex, semáforos, variables condicionales [11], etc.).

Nos encontramos por tanto con secciones de código susceptibles a realizar cambios que provoquen una inconsistencia sobre los datos (secciones críticas) y con un conjunto de procesos que, en pro de la eficiencia, accederán a dichos datos sin el criterio que nuestra aplicación requiera (condiciones de carrera).

3.1.3. FRAMEWORK PARA APLICACIONES WEB

Un framework para aplicaciones web es una estructura de apoyo o soporte, a través de la cual otro proyecto de software puede ser organizado y desarrollado. Puede entenderse como una aplicación genérica e incompleta a la que podemos añadir la lógica de nuestras necesidades para desarrollar así una aplicación concreta.

Este tipo de framework intenta reducir los esfuerzos del desarrollador a la hora de diseñar e implementar una aplicación o servicio web. Esto es posible debido a que las actividades necesarias para el desarrollo de una aplicación web, están muy delimitadas y suelen repetirse con frecuencia, lo que beneficia la reutilización de código. Las principales ventajas obtenidas al utilizar un framework de este tipo son:

- Menor esfuerzo y rapidez en el desarrollo.
- Desarrollo estructurado en torno a una estructura fija.
- Reutilización de código existente (no se debe reinventar la rueda).
- El diseño es más sencillo debido a que se centra directamente en la solución al problema.

Aunque varía en función del framework utilizado, las principales características que suelen incluir son:

- **Estructuras para plantillas:** Se ofrecen plantillas que separan la parte estática de la dinámica en las diversas páginas web. En ocasiones es necesario especificar únicamente la parte dinámica de dichas páginas.
- **Control caché:** Para disminuir el ancho de banda, la carga del servidor y las altas latencias, se almacenan respuestas en caché que son gestionadas de forma automática.
- **Seguridad:** Se conceden mecanismos de seguridad para gestionar la autenticación y permisos de los clientes que accedan a la aplicación web.
- **Base de datos:** Incluyen librerías que permiten el uso de la base de datos. En algunos casos se ofrecen mecanismos de mapeado de objetos con registros de la base de datos, llevados a cabo mediante capas de persistencia (Hibernate [12]).

3.1.4. ANÁLISIS DE FRAMEWORK PARA DESARROLLO WEB

A continuación se detallan las características de los diversos frameworks de desarrollo web existentes en el mercado y que han sido estudiados de manera previa a la elección de uno de ellos. Utilizado como apoyo a la implementación no se realizará un análisis intensivo del mercado actual, por lo que se evaluarán las características de un pequeño grupo de frameworks.

J2EE

Características

Framework desarrollado por SUN que ofrece las mejores perspectivas en el desarrollo de aplicaciones web basado en software libre. Sus principales características son:

- **Multiplataforma:** Al estar desarrollado en Java, es posible desarrollar aplicaciones web para cualquier sistema operativo que soporte dicho lenguaje.
- **Constante Evolución:** Un conjunto de más de 500 importantes empresas controlan la evolución de este framework.
- **Competitividad:** El framework permite definir soluciones propias que ofrezcan características diferentes respecto a eficiencia, rendimiento, precio, etc.
- **Madurez:** Dispone de un gran arraigo en el mercado y multitud de proyectos realizados con dicho framework.
- **Software Libre:** Todo el desarrollo se realiza utilizando software libre.

Licencia

Licencia gratuita y libre.

Lenguaje de programación

Únicamente Java.

ASP.NET

Características

Desarrollado por Microsoft como sucesor a la tecnología **ASP**. Sus características más distintivas son:

- **Multilenguaje:** Es posible desarrollar una misma aplicación web usando varios lenguajes de programación de forma simultánea.
- **Integración:** El framework obtiene el mejor rendimiento de entornos con software y hardware dependiente de Microsoft.
- **Visual Studio:** Se pone a disposición del desarrollador esta potente herramienta que ofrece un entorno homogéneo de trabajo.
- **No requiere experiencia:** Debido a que es posible utilizar lenguajes como **Visual Basic,** que facilitan enormemente la implementación, no son necesarios programadores muy experimentados.

Licencia

Licencia propietaria.

Lenguaje de programación

Compatible con Visual Basic, C# y **J#**.

3.1.5. XML

Desarrollado por el World Wide Web Consortium, **XML** es un lenguaje de marcas que deriva del lenguaje **SGML**. Este lenguaje se propone como un estándar para el intercambio de información estructurada entre plataformas distintas. Se trata de una tecnología sencilla de gran relevancia en la actualidad, ya que solventa problemas de compatibilidad entre plataformas a la hora de compartir información de una forma segura, fiable y fácil.

Actualmente es considerado un estándar aceptado y aprobado por la industria, aunque ha recibido numerosas críticas debido a su nivel de detalle y complejidad en determinados casos (el mapeo puede llegar a ser un problema).

Las ventajas que aporta el uso de este metalenguaje son las siguientes:

- **Extensible:** Una vez diseñado e implementado, es posible extender el lenguaje añadiendo nuevas etiquetas sin que esto ocasione un problema.
- **Analizador estándar:** No es preciso definir un analizador para cada versión del lenguaje lo que acelera el desarrollo.
- **Compatibilidad:** Es posible comunicar plataformas distintas utilizando este lenguaje, sólo es necesario entender su estructura para poder procesarla.
- **Flexibilidad:** Al añadir un significado y asociarle un contexto a los datos, obtenemos flexibilidad en la estructuración de documentos.

La estructura de los documentos XML es arbórea y está compuesta por fragmentos de información. Se intenta de esta forma obtener abstracción y reutilización de la información. Para ello, el lenguaje permite definir sobre la información los siguientes elementos:

- **Etiquetas:** Marcas que indican el principio y fin de la información que conforma un elemento. La información puede ser vacía o contener etiquetas que definan nuevos sub elementos.
- **Atributos:** Propiedades definidas para cada uno de los elementos de un documento. La información que contengan va entrecomillada.
- **Secciones CDATA:** Permiten especificar datos empleando cualquier carácter (elimina problemas con caracteres especiales).

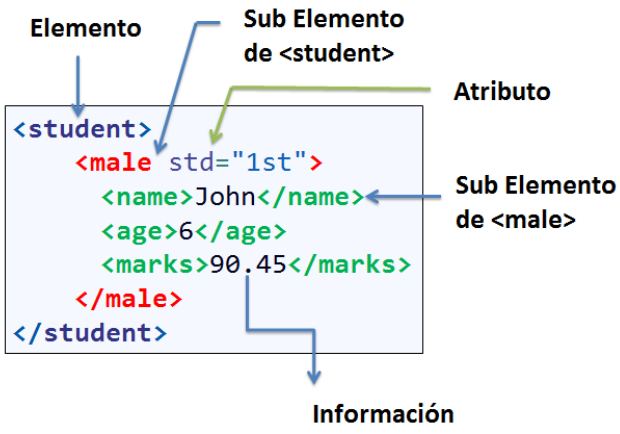


Ilustración 12: Ejemplo de XML

Los documentos formados por XML pueden ser analizados para comprobar si su estructura sintáctica es correcta, lo que denominaremos como bien formado. La estructura sintáctica y los elementos que puede contener un determinado tipo de documento, se recogen en los documentos denominados **DTD**.

3.1.6. SEGURIDAD

La seguridad orientada a las comunicaciones debe cumplir con las tres exigencias de cualquier sistema seguro que englobemos en el marco de la informática:

- **Confidencialidad:** La información comunicada debe ser accesible únicamente por las entidades autorizadas.
- **Integridad:** La modificación de la información transmitida debe poder realizarse únicamente por las entidades autorizadas.
- **Disponibilidad:** La información autorizada debe estar disponible siempre que las entidades con autorización la soliciten.

Además, la seguridad en las comunicaciones involucra otros aspectos de manera adicional a los anteriormente descritos:

- **Autenticación:** Se debe poder identificar al propietario o emisor de la información.
- **No repudio:** No debe ser posible que las diversas entidades nieguen la procedencia de los datos emitidos o la recepción de los mismos.

Actualmente la seguridad informática provee de algoritmos, funciones, protocolos, arquitecturas y mecanismos para satisfacer todas las exigencias mencionadas.

3.1.6.1. CIFRADORES

El cifrador es un elemento utilizado en la seguridad informática para satisfacer la necesidad de confidencialidad de la información. Conceptualmente puede entenderse como una caja negra a la que se le suministra el mensaje en claro que queremos cifrar y una clave, y del que obtenemos el mensaje cifrado. Es requisito indispensable que todo cifrador disponga de un descifrador asociado, con el que inversamente pueda calcularse el mensaje en claro a partir del mensaje cifrado y la clave.

La seguridad de cualquier cifrador o algoritmo de cifrado, debe residir únicamente en el secreto de la clave, por lo que el funcionamiento de dichos algoritmos podría ser público y no pondría en peligro la seguridad de la información cifrada mediante uno de ellos.

Atendiendo al tipo de algoritmo, podemos encontrarnos los siguientes tipos:

- **Cifrado de flujo:** El cifrado se realiza bit a bit con claves de gran longitud. Estas claves pueden ser de un solo uso o ser generadas mediante generadores de claves pseudoaleatorias.
- **Cifrado de bloque:** El cifrado se realiza sobre conjuntos de bits de la misma longitud.

A su vez, los algoritmos de cifrado de bloque pueden ser catalogados atendiendo al tipo de clave utilizada:

- **Simétricos:** La clave de cifrado y descifrado son idénticas, por lo que ambas partes de la comunicación deben compartirla.
    - **Ventajas:**
        - Algoritmos más eficientes y fácilmente implementables en hardware.
        - Claves de menor tamaño que las asimétricas.

    - **Desventajas:**
        - Distribución de clave costosa debido a la inseguridad de los canales.
        - Se requiere un gran número de claves (una por cada pareja de entidades).
- **Asimétricos:** Las claves de cifrado y descifrado son distintas. Se utilizan pares de claves para cada una de las entidades involucradas (una clave pública utilizada para descifrar y una clave privada utilizada para cifrar). Al ser las claves públicas de dominio público, es necesario que sea computacionalmente inabordable calcular por medio de estas últimas las claves privadas. Este tipo de cifrado también proporciona autenticación ya que el emisor del mensaje puede ser únicamente el poseedor de la clave privada.
    - **Ventajas:**
        - Distribución de claves sencilla.
        - Se requiere un menor número de claves (una pareja por cada entidad).
    - **Desventajas**:
        - Algoritmos menos eficientes.
        - Claves de mayor tamaño que las simétricas.

Debido a las deficiencias mostradas por ambos sistemas de cifrado, la mejor solución y la que actualmente se utiliza consiste en intercambiar una clave simétrica cifrada mediante un algoritmo asimétrico. Una vez se comparte la clave, se utilizan algoritmos simétricos para realizar el cifrado de mensajes.

3.1.6.2. PRINCIPALES CIFRADORES DE BLOQUE

Estos cifradores actúan sobre grupos de símbolos de tamaño fijo (en contraposición a los cifradores de flujo que actúan sobre símbolos aislados). Los bloques o grupos de símbolos son independientes entre ellos, por lo que cifrado y descifrado puede realizarse de forma asilada al resto de bloques.

A continuación se recogen y detallan las características de los cifradores más conocidos y utilizados:

DES

- Puede ser implementado en software y hardware (más rapidez).
- Utiliza bloques de 64 bits, claves de 48 bits (64 bits la inicial) y realiza 16 iteraciones.
- Utiliza los principios de **Difusión** y **Confusión**.
- Existencia de claves débiles o semidébiles (generan las mismas claves o únicamente dos distintas).
- Tamaño de clave pequeño (vulnerable). Resuelto mediante **Triple DES**.
- Seguridad de las **S-Cajas** sin demostrar.
- Vulnerable a ataques de **Criptoanálisis diferencial** (requiere grandes cantidades de texto escogido y su cifrado).
- Utilizado para información con caducidad a corto plazo.

IDEA

- Fácilmente implementable en software y hardware.
- Utiliza bloques de 64 bits, sub claves de 16 bits (128 bits la inicial) y realiza 8 iteraciones.
- Utiliza los principios de difusión y confusión.
- Seguridad demostrada mediante justificación matemática.

RC5

- Número de iteraciones, tamaño de bloque y longitud de clave variable (escogidas por el usuario).
- Realiza tres operaciones inversibles (suma módulo 2^w, **XOR** bit a bit y rotación a izquierda).
- Vulnerable debido a la debilidad de las claves generadas.

RIJNDAEL

- Utiliza longitudes de clave y bloque variables a lo largo de las iteraciones (entre 128 y 256 bits).
- El número de iteraciones es variable y emplea funciones inversibles de **Transposición** y **Sustitución**.
- Fácilmente paralelizable (mayor eficiencia).
- El descifrado se realiza con las funciones inversas aplicadas en orden inverso al de cifrado.
- Robusto ante ataques exhaustivos, criptoanálisis línea y diferencial.
- No posee claves débiles ni semidébiles.

3.1.6.3. FUNCIONES RESUMEN

La seguridad informática provee un conjunto de algoritmos denominados funciones resumen, que con frecuencia se utilizan para garantizar la integridad de la información. Estas funciones están destinadas a transformar un mensaje o conjunto de símbolos en otro menor (de tamaño fijo) que contiene información del mensaje original y es difícilmente falsificable.

Una función resumen debe cumplir exhaustivamente las siguientes propiedades:

- La salida de la función resumen debe ser de longitud fija (recomendablemente mayor que 128 bits).
- No tiene que ser costoso obtener el resumen de un mensaje en claro.
- Tiene que ser computacionalmente intratable calcular el mensaje a partir de su resumen.
- Tiene que ser computacionalmente intratable obtener dos mensajes que produzcan el mismo resumen.
- Tiene que ser computacionalmente intratable encontrar dos mensajes aleatorios que produzcan el mismo resumen. Suelen añadir información de la longitud del mensaje para evitar que dos mensajes de distinta longitud produzcan el mismo resumen.

Debido a que el conjunto de resúmenes que una función puede generar está limitado por la longitud fija de salida de dicho algoritmo y que el conjunto de mensajes de entrada es infinito, existen por tanto mensajes distintos que produzcan los mismos resúmenes (de hecho, existen infinitos mensajes que produzcan los mismos resúmenes).

La fortaleza de una función resumen reside en menor medida en evitar colisiones (encontrar dos mensajes que produzcan un mismo resumen) y en mayor medida en evitar ataques de pre imagen (calcular un mensaje que produzca el mismo resumen que otro texto escogido).

3.1.6.4. PRINCIPALES FUNCIONES RESUMEN

Aunque existe un mayor número de funciones resumen, este apartado tratará únicamente los dos más conocidos y utilizados algoritmos de funciones resumen:

MD5

- La función utiliza bloques de 512 bits de entrada y produce una salida de 128 bits.

- Hace uso de 4 registros de 32 bits con valor inicial constante.
- Interpreta el mensaje como **Little Endian**.

SHA-1

- Utiliza bloques de 512 bits y produce una salida de 160 bits.
- Hace uso de 5 registros de 32 bits con valor inicial constante.
- Interpreta el mensaje como **Big Endian**.

3.1.6.5. PRINCIPALES PROTOCOLOS DE INTERCAMBIO DE CLAVE

Pueden ser definidos como protocolos criptográficos en los que se establece una secuencia de acciones entre dos o más entidades, para poner en común entre estas últimas una información secreta y compartida. Esta información suele ser una clave que después se usará en algún cifrador para establecer una comunicación segura. La definición de estos protocolos surge de la necesidad de enviar una clave utilizando un canal inseguro.

Los principales protocolos de intercambio de clave se exponen a continuación:

DIFFIE-HELLMAN

Protocolo utilizado con frecuencia para acordar claves simétricas que posteriormente se utilizarán para cifrar una sesión. Se trata de un algoritmo no autenticado, aunque se utilice como base de otros protocolos de autenticación.

La seguridad del algoritmo reside en la extrema dificultad computacional de calcular logaritmos discretos [13], aunque es vulnerable a ataques del tipo Man in the middle [31].

ELGAMAL

Este protocolo provee negociación de clave en un único paso y además autenticación del receptor hacia el emisor, siempre que la clave pública del receptor sea previamente conocida por el emisor.

Al igual que el anterior, su seguridad reside en la dificultad de calcular logaritmos discretos.

3.4. PERSISTENCIA DE DATOS

3.4.1. BASES DE DATOS

Una de las principales problemáticas de la mayoría de los sistemas informáticos es la de almacenar un gran número de datos de forma persistente. La solución ante este problema requiere de un soporte físico y de mecanismos que permitan controlarlo y ordenarlo bajo las necesidades del desarrollador. Estas necesidades son la causa de que aparezca el concepto de las bases datos.

Una base de datos puede describirse como un almacén de datos que guardan relación entre ellos y que se encuentran organizados. Conceptualmente una base de datos representa una porción del mundo real que el diseñador considera relevante para su sistema y que puede usarse para un propósito. Los datos almacenados deben ser fácilmente accesibles y manipulables con ayuda de la base de datos.

El empleo de bases de datos conlleva una serie de ventajas y desventajas que se analizan a continuación:

- **Ventajas**
    - **Redundancia:** No se almacenan datos repetidos como sucede en los sistemas de ficheros convencionales. Es posible encontrar redundancia en una base de datos aunque únicamente para modelar relaciones entre datos.
    - **Consistencia:** Debido a que todas las redundancias de datos están controladas por la base de datos, cualquier cambio supone una actualización de las referencias, manteniendo así su consistencia.
    - **Integridad:** Existen mecanismos que evitan que se violen determinadas reglas que supongan poner en riesgo la integridad de los datos.
    - **Estandarización:** Los datos manejados por la base de datos están almacenados de acuerdo a unos estándares que facilitan su intercambio.
    - **Compartición:** Todos los usuarios con autorización compartirán los datos almacenados, lo que supone un ahorro respecto a espacio físico.
    - **Seguridad:** Establece un control de usuarios, grupos y permisos que conceden seguridad y evitan vulnerabilidades. Además incluyen mecanismos que facilitan la creación de copias de seguridad que puedan ser recuperadas ante un error o pérdida de datos.
    - **Acceso:** Definen lenguajes de consultas que permiten a sus usuarios acceder a los datos de una forma sencilla.
    - **Mantenimiento:** Al separar la lógica de los datos de las aplicaciones que los gestionan, un cambio en los datos no tiene que suponer necesariamente un cambio en las aplicaciones.
    - **Productividad:** Se ofrecen al programador multitud de funcionalidades que supondrían una gran inversión de tiempo y dinero si se realizasen sobre un sistema de ficheros convencional.
    - **Concurrencia:** El acceso a las bases de datos está gestionado de forma que si varios usuarios acceden a un mismo dato, no se producen errores, fallos ni interferencias.
- **Desventajas**
    - **Equipamiento adicional:** Las necesidades de procesado y almacenamiento se ven incrementadas al hacer uso de una base de datos. La implantación de una base de datos supone un coste adicional alto.
    - **Complejidad:** Las bases de datos son sistemas complejos que conceptualmente son difíciles de comprender.

- **Vulnerabilidades:** Centralizar los datos hace que estos sean más vulnerables a fallos.

Existen diversos tipos de bases de datos en función del modelo de administración de datos utilizado, a continuación se detallan los principales:

- **Bases de datos transaccionales:** Su finalidad es el envío y recepción de datos a grandes velocidades, su uso suele ser poco común y reducido al entorno de análisis de calidad.
- **Bases de datos jerárquicas:** Sus datos se almacenan en torno a una jerarquía en la que cada dato puede mantener relación con un único dato **padre** (en caso de no tener padre, se denomina raíz) y con varios datos **hijos** (en caso de no tener descendencia, se denominan hojas). Este tipo de base de datos es útil cuando existe una gran cantidad de datos que requieren un acceso en un tiempo reducido o establecido.
- **Bases de datos de red:** Similar al jerárquico sin la restricción de que un nodo **hijo** deba tener únicamente un **padre**.
- **Bases de datos relacionales:** Es el tipo más común y utilizado en el que la idea fundamental es el uso de las relaciones. Estas relaciones podrían considerarse en forma lógica como conjuntos de datos llamados **tuplas**. Cada relación se concibe cómo una tabla que está compuesta por registros (las filas de una tabla) que representarían las **tuplas**, y campos (las columnas de una tabla). La información puede ser recuperada o almacenada mediante consultas que ofrecen una amplia flexibilidad.
- **Bases de datos orientadas a objetos:** Propias de sistemas informáticos orientados a objetos que pueden ser almacenados independientemente de su complejidad y de cualquier tipo de asociación existente (encapsulación, herencia o polimorfismo).

El mantenimiento de una base de datos puede realizarse de diferentes maneras. Por una parte es posible mantenerlas de forma manual, en la que el propio usuario es el encargado del control, o pueden utilizarse un conjunto de aplicaciones que se encargan de realizar las gestiones requeridas. Este conjunto de aplicaciones adoptan el nombre de **SGBD**, mediante los cuales se definen, construyen y manipulan las bases datos manteniendo su integridad y ausencia de redundancia.

3.4.1.1. SQL

Para poder realizar una comunicación entre los diferentes usuarios o aplicaciones y la base de datos, es necesario definir un lenguaje mediante el cual se efectúen inserciones, modificaciones o consultas. Para evitar la definición de infinidad de estos lenguajes, debido al gran número de bases de datos existentes, se hace uso de estándares que permiten realizar estas operaciones de forma universal.

Se define de esta forma el lenguaje estándar conocido como SQL que permite comunicar cualquier tipo de aplicación (PHP, Java, ASP, etc.) con cualquier base de datos (Oracle, MySQL, Microsoft Access, etc.). Aunque se trate de un estándar, cada base de datos amplía su propio lenguaje SQL con funciones y características propias que lo adaptan a la estructura y funcionamiento de su base de datos.

Este potente leguaje de fácil aprendizaje, dispone de dos partes bien diferenciadas:

- **Lenguaje de definición de datos:** Encargado de la modificación de la estructura de los objetos de la base de datos.
- **Lenguaje de manipulación de datos:** Destinado a la gestión de la base de datos, lo que incluye consultas, manipulación de datos, organización por modelos, etc.

3.4.1.2. TRANSACCIONES

Desde el punto de vista de la base de datos, las operaciones que el usuario realiza están compuestas a su vez por varias operaciones elementales. De esta forma, lo que el usuario ve como una transferencia bancaria, la base de datos lo interpreta como restar fondos a una cuenta y añadirlos a otra. Se denomina transacción al conjunto de operaciones que forman una unidad lógica que accede y/o actualiza elementos de datos.

A continuación se enumeran las características de las transacciones y la responsabilidad del cumplimiento de cada una de ellas:

- **Atomicidad:** Es necesario que se realicen todas las operaciones del conjunto o que no se realice ninguna. No se permiten ejecuciones parciales. Se encuentra bajo la responsabilidad de la base de datos.
- **Consistencia:** La ejecución de la transacción debe conservar la consistencia de la base de datos. Es responsabilidad del programador que codifica la transacción.
- **Aislamiento:** Se garantiza cada transacción, ignorando el resto en una ejecución concurrente de transacciones. Es responsabilidad de la base de datos.
- **Durabilidad:** Los cambios realizados en la base de datos se mantienen tras la finalización de una transacción. Se encuentra bajo la responsabilidad de la base de datos.

Las bases de datos ofrecen mecanismos para indicar en el tiempo, en inicio y fin de una transacción. El fin de la transacción debe indicar si los cambios realizados en la base de datos se consideran consistentes o no, para esto se ofrecen las siguientes funciones:

- **Commit:** Se utiliza para actualizar finalmente el estado de la base de datos.
- **Rollback:** Se utiliza para deshacer los cambios realizados durante la transacción.

3.4.1.2.1. CONCURRENCIA

Las bases de datos ofrecen además mecanismos para solventas los problemas clásicos de concurrencia que puedan ocasionarse durante la ejecución de transacciones. Esto se debe a que en ocasiones los mecanismos clásicos de control de concurrencia ofrecidos por el propio sistema operativo o por librerías específicas, no son suficientes para gestionar la concurrencia que provoca el uso de una base de datos.

Entre todas las ofrecidas, pueden catalogarse en dos tipos principales:

- **Técnicas pesimistas:** Previenen ante un error antes de que se produzca.
    - **Bloqueo:** Se asocia una variable (cerrojo) a cada elemento de datos que describe el estado de dicho elemento respecto a las operaciones que se pueden realizar sobre él. Podemos encontrar bloqueos exclusivos (un único bloqueo por recurso) o compartidos (varios bloqueos por recurso). Ocasionan problemas de interbloqueo.
    - **Marcas de tiempo:** Mediante las cuales se determina el orden de las transacciones. Evitan interbloqueos.
- **Técnicas optimistas:** Corrigen el error en caso de que se produzca.
    - **Técnicas de validación:** Se mantiene una copia de los datos para cada usuario de forma que no se modifiquen los datos durante la lectura. Mediante funciones de commit se hacen los cambios persistentes y por tanto visibles a todos los usuarios.

3.4.2. ANÁLISIS DE LOS SISTEMAS GESTORES DE BASES DE DATOS

Los SGBD son un tipo de software de carácter muy específico que sirve de interfaz entre el usuario, la base de datos y las aplicaciones que hacen uso de ella. Principalmente está compuesta por un lenguaje de definición de datos, de un lenguaje de manipulación de datos y de un lenguaje de consulta.

Existen multitud de alternativas para diseñar, crear y mantener una base de datos. Entre las distintas alternativas es preciso realizar una catalogación de importancia entre las que son de distribución libre y las que su uso supone un coste adicional. Debido a la temática y propósito del proyecto aquí tratado, que se aleja del aprendizaje y uso de bases de datos, se procederá detallando brevemente las alternativas estudiadas para su posterior uso.

POSTGRESQL

Características

Probablemente una de las bases de datos más avanzadas de código abierto. Reúne una gran cantidad de características y funciones que le han hecho ganar fuerza incluso en el mundo empresarial, llegando a rivalizar incluso con bases de datos comercializadas como Oracle.

Entre sus características más relevantes se encuentran:

- **Alta concurrencia:** Es posible acceder a una tabla mientras se está escribiendo sin provocar inconsistencias.
- **Tipos nativos:** Soporta gran variedad de tipos que no son soportados por otras bases de datos tales como: Direcciones IP, direcciones MAC, figuras geométricas, etc.
- **Transacciones:** Ofrece integridad transaccional y soporte para transacciones distribuidas.

Licencia

Licencia gratuita y libre **BSD**.

MYSQL

Características

Normalmente asociada con bases de datos de pequeñas y medianas empresas, su gran popularidad surgió con el uso de la misma para la mayoría de aplicaciones web. Su manejo, configuración y consumo de recursos, la convierten en una base de datos realmente cómoda.

Los rasgos más característicos de esta base de datos son los siguientes:

- **Compatibilidad:** La inmensa mayoría de sistemas y plataformas son soportados.
- **Motores de almacenamiento:** Es posible definir que motor de almacenamiento será empleado con cada una de las tablas. Cada uno de estos motores está diseñado para incrementar la eficiencia en función del uso que se vaya a dar a cada tabla.
- **Transacciones:** Es posible agrupar transacciones indistintamente de la conexión a la que pertenezcan para aumentar su eficiencia.

Licencia

Bajo la licencia **GNU GPL**, se obliga a que cualquier distribución privada se haga bajo esa misma licencia.

ORACLE

Características

Las empresas caracterizadas por su gran volumen de información hacen uso de una base de datos Oracle. Actualmente se encuentra ampliamente implantada en el mundo empresarial con un dominio en el mercado sobre sus competidores y es considerado como uno de los sistemas de bases de datos más completos.

Sus rasgos más característicos son:

- **Multiplataforma:** Soportada en las principales plataformas del mercado.
- **Estabilidad:** La recuperación ante errores y la integridad y consistencia de los datos se encuentra completamente asegurada para esta base de datos.
- **Escalabilidad:** Se trata de la base de datos con mayor eficiencia ante un crecimiento establecido.
- **Transacciones:** Posiblemente Oracle ofrezca uno de los sistemas más eficientes y seguros de transacciones.

Licencia

Es necesario adquirir una licencia privada de la base de datos para poder utilizarla en cualquier tipo de desarrollo software.

3.5. MODELADO 3D

El modelado 3D representa formas y objetos de geometría espacial mediante una colección de vértices unidos por aristas, las cuales delimitan porciones planas del espacio llamadas polígonos. Utilizando estas 3 primitivas geométricas (vértices, aristas, polígonos) y efectuando operaciones matemáticas sobre ellas, es posible modelar cualquier tipo de objeto.

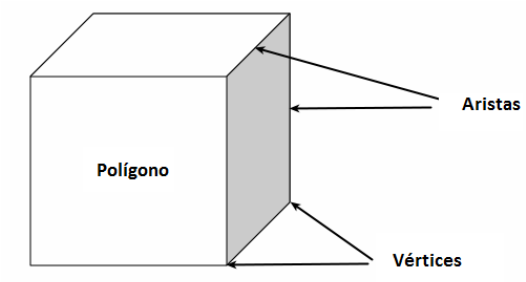


Ilustración 13: Primitivas geometría

Los vértices son simples coordenadas en el espacio y las conexiones entre ellos generan objetos, por lo que cualquier modelo puede resumirse matemáticamente.

3.5.1. TÉCNICAS DE MODELADO 3D

Existen diversas técnicas de modelado que pueden ser usadas en función de las necesidades del diseñador. Por lo general, el modelado se lleva a cabo como una combinación entre varias de estas técnicas:

- **Composición de primitivas geométricas:** Figuras geométricas básicas (cubos, cilindros, esferas o conos) se agrupan para formar figuras más complejas. Mediante transformaciones geométricas básicas (translación, escalado y rotación) y operaciones booleanas (intersección, unión y diferencia) es posible distribuir las distintas figuras básicas hasta obtener el modelo deseado.
- **Mallas de polígonos:** Modelado por medio de transformaciones de los vértices, aristas y polígono de una malla de celdas rectangulares o triangulares que da lugar a modelos complejos. Aunque permiten un modelado más personal, las curvas o bordes suavizados incrementan la dificultad al tener que realizarse manualmente.
- **Extrusión:** Consiste en tomar una forma geométrica plana y proyectarla en su tercera dimensión restante. De esta forma un círculo permite generar un cilindro o una corona circular genera un tubo. El modelo matemático **NURBS**, permite realizar extrusiones sobre curvas definidas por el diseñador y girando en torno a un eje. Las posibilidades de este último modelo son infinitas, ya que es posible modelar mediante formas definidas tridimensionales que conforman los límites de las curvas que lo forman.

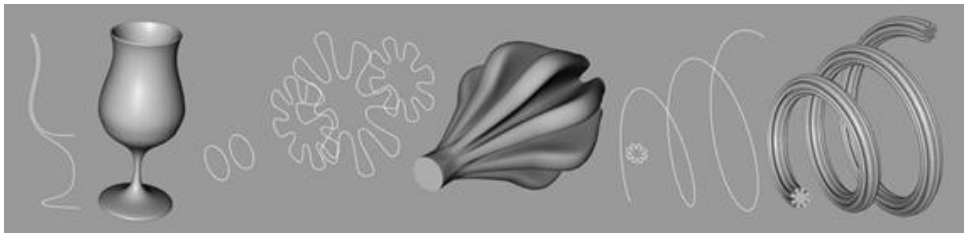


Ilustración 14: Aplicaciones de NURBS

3.5.2. MATERIALES Y TEXTURAS

El renderizado de una escena requiere que todos los objetos que se encuentran en ella, dispongan de un material asociado. El material agrupa numerosos atributos del objeto a representar, tales como: Textura (imagen que recubre al objeto), color, brillo, grado de reflexión y refracción, color reflejado para la luz ambiente, etc.

La textura se extiende a lo largo de la superficie del objeto respetando el mapeado UV [15]. Con él es posible definir la correspondencia entre los pixeles de una textura y los vértices del objeto tridimensional. Adicionalmente existen diversas técnicas que complementan las texturas anteriormente descritas:

- **Bump mapping:** Permite representar la superficie de un objeto previamente iluminada, con aspecto rugoso, poroso o con relieve sin necesidad de alterar su geometría. Las texturas asociadas a esta técnica son imágenes en escala de grises cuyos valores representan la profundidad de los vértices representados.
- **Normal mapping:** Variante de la anterior donde mediante mapas de colores RGB se define la posición de los vértices en tres dimensiones. Al poseer más información para representar el objeto, la sensación de realismo es mayor.
- **Parallax mapping:** Evolución de las dos técnicas anteriores que desplaza en cada punto de un polígono las coordenadas de la textura en función del ángulo que forma el vector vista (dirección de la cámara) con respecto a las normales de la superficie del objeto.

Ilustración 15: Ejemplo de Parallax mapping

- **Stencil mapping:** Consiste en aplicar varias texturas sobre una misma superficie que mediante plantillas, definen que texturas se superponen a cuales. Esta técnica suele utilizarse para representar terrenos que requieran varias texturas (por ejemplo, un camino de tierra sobre un terreno de hierba).
- **Light mapping:** De forma similar a la anterior, se definen mediante plantillas las zonas que serán iluminadas con mayor o menor brillo de una superficie.

3.5.3. MODELADO PROCEDURAL

En ocasiones, el modelado realizado por el propio diseñador no es viable o una alternativa para la solución requerida. Esto puede deberse a que por cuestiones de tiempo, el modelado suponga un trabajo inabordable o porque sea requisito indispensable que ciertos modelados deban ser auto generados por la aplicación.

La solución ante uno de estos escenarios consiste en generar computacionalmente esos modelos, lo denominado modelado procedural. Existen multitud de técnicas usadas para generar modelos 3D y texturas a partir de un conjunto de reglas. Este conjunto de reglas suele encontrarse dentro de un algoritmo y es configurable mediante parámetros.

Existen diversas funciones matemáticas cuyo desarrollo y posterior aplicación, generan estructuras que se asemejan a modelados usados con frecuencia y que suelen guardar relación con elementos de la naturaleza (plantas, terrenos, montañas, nubes, etc.).

A continuación se exponen algunas de las funciones y algoritmos más usados y su aplicación en el modelado:

- **Funciones de Ruido:** Función matemática que construyen un valor que varía seudo aleatoriamente en el espacio o tiempo. La más famosa y utilizada es el Ruido de Perlin [16], similar al ruido blanco, es utilizado para modelar terrenos, construir superficies y representar efectos como fuego, humo, nubes, etc.
- **Sistemas L:** Gramática formal que se utiliza para modelar el crecimiento de elementos de la naturaleza tales como plantas u organismos. Su naturaleza recursiva hace que sus construcciones dispongan de **Autosemejanza**, propiedad que comparten elementos como árboles, vegetales, algas, etc.
- **Fractales:** Objetos cuya estructura está compuesta por la repetición de un elemento básico a diferentes escalas. Existen multitud de fractales en la naturaleza que han sido

matemáticamente definidos y cuyo uso permite modelar elementos como paisajes, terrenos y vegetación.

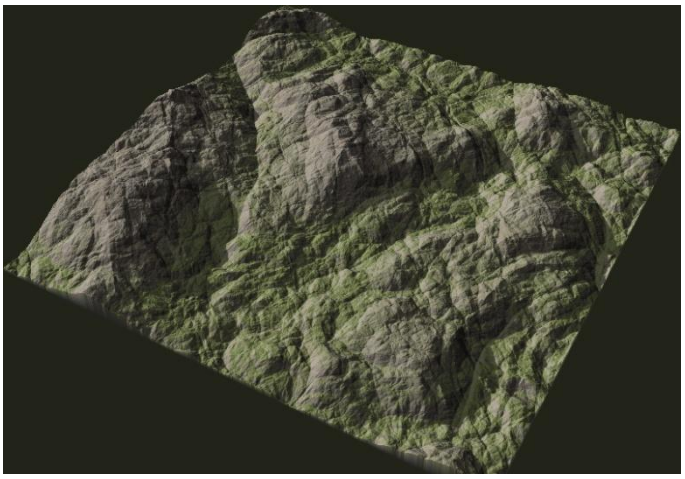


Ilustración 16: Aplicación del Ruido de Perlin en terrenos

Todas las técnicas aquí expuestas comparten dos propiedades fundamentales. Por una parte el desarrollo de los algoritmos que definan este tipo de funciones es sencillo, ya que no requieren de un notorio esfuerzo por parte del programador, exceptuando quizá la comprensión conceptual que requieren. Por otra parte la ejecución de este tipo de funciones requiere de una capacidad de cómputo considerable y que debe de tenerse en cuenta a la hora de escoger una solución que adopte una de estas técnicas.

3.5.4. ILUMINACIÓN

Todas las texturas y materiales anteriormente descritos no serían visibles mientras no existan fuentes de luz en una escena a *renderizar*. Las fuentes de luz pueden ser de varios tipos:

- **Luz posicionada en un punto sin dirección:** Emite en todas direcciones.
- **Luz posicionada en un punto con dirección**: Emite un cono de rayos de luz en una dirección determinada.
- **Foco de luz:** De forma circular o cuadrada que emite rayos de luz paralelos desde la superficie del mismo.
- **Luz solar:** Representada como rayos paralelos de luz en una determinada dirección y sin origen visible (origen en el infinito).
- **Luz ambiente:** Luz suave global no proporcionada desde ninguna fuente concreta.

También cabe destacar que en el caso de las luces emitidas desde un punto o desde los focos, la distancia que alcanzan los rayos luz es configurable, quedando todo aquello que permanezca fuera de esa distancia progresivamente oscurecido, a diferencia de la luz solar o ambiente que llegan a todas partes.

3.5.5. SHADERS

Conjunto de instrucciones que se ejecutan en la **GPU**. Escritos en lenguaje de definición de shaders, se utilizan para aprovechar la aceleración del hardware disponible en la GPU durante la fase de renderizado. Su uso suele verse reducido a realizar transformaciones y al a crear efectos de iluminación, fuego, niebla, etc.

Aunque en sus inicios los shaders eran definidos mediante instrucciones de ensamblador o con lenguajes de bajo nivel, la constante evolución de esta tecnología ha llevado a la actual existencia de diversos lenguajes de definición de shaders de alto nivel (**HLSL**, **GLSL**, etc.).

Es posible suministrar información al shader desde la aplicación por medio de parámetros. Aunque los tipos de datos aceptados son limitados, es posible asignarle elementos como coordenadas, colores, texturas, etc. El funcionamiento de los shaders se divide en la ejecución de tres funciones de parámetros limitados que se ejecutan una vez por cada pixel mostrado y renderizado por la GPU:

1. **VertexShader:** El cometido de esta función es la transformación de la posición final del vértice.
2. **GeometryShader:** Su cometido es el de generar nuevas primitivas de forma dinámica o de modificar las existentes.
3. **PixelShader:** En esta última fase se procede a realizar transformaciones sobre los pixeles finales mostrados al usuario (puntos en la pantalla de coordenadas bidimensionales). Su cometido es el de modificar la luz, color, profundidad, etc.

Dependiendo de los lenguajes de definición, estas funciones pueden variar en nombre y número, ya que es posible que se unifiquen o dividan en más cada una de las aquí mencionadas.

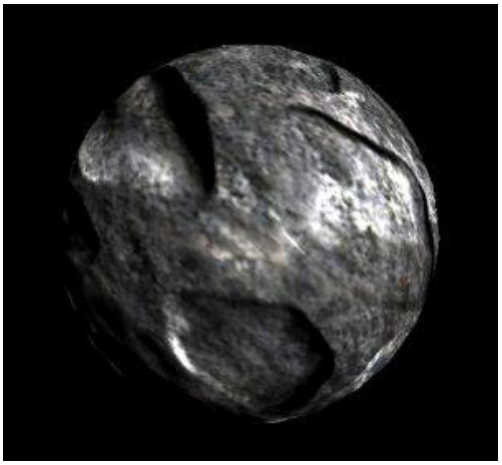


Ilustración 17: Esfera tras aplicarle shaders

3.5.5.1. FILTROS DE POST-PROCESADO

Son shaders aplicados a la imagen renderizada y aplicados antes de ser mostrada al usuario. Su finalidad es la de aplicar también efectos visuales aunque a diferencia del renderizado, es preciso disponer de la imagen completa para poder ser aplicado.

Basados en la ejecución del VertexShader y PixelShader anteriormente mencionados, pueden suponer una reducción importante de la tasa de **FPS**, ya que cada fotograma mostrado sufre un procesado adicional.

Existen diversidad de filtros y shaders ya definidos que permiten realizar efectos sobre nuestras aplicaciones tridimensionales. Suele ser frecuente que dichos efectos sean proporcionados por el propio motor gráfico.

Ilustración 18: Aplicación del filtro Toon-Shading en el juego Jet Set Radio

3.6. LIBRERÍAS DE DESARROLLO DE VIDEOJUEGOS

3.6.1. LIBRERÍA DE DESARROLLO DE VIDEOJUEGOS

Una librería de desarrollo o lo que comúnmente se conoce como API, es una herramienta software que facilita la implementación de aplicaciones. Implementadas como aplicaciones y/o bibliotecas, ofrecen una interfaz que puede ser utilizada en otra aplicación y que suele hacer uso de llamadas al sistema operativo o de elementos propios como estructuras de datos, funciones, rutinas, protocolos, clases, etc.

Una librería de desarrollo de videojuegos ofrece al desarrollador una capa de abstracción sobre la que apoyarse para diseñar e implementar un videojuego. Este tipo de librearías abstrae elementos que son comunes en la mayoría de videojuegos, tales como herramientas elementos gráficos, de físicas y de interacción entre otros.

3.6.2. MOTOR DE JUEGO

Normalmente presentado como una aplicación y/o bibliotecas, son un conjunto de funciones que son utilizadas para diseñar, implementar y ejecutar un videojuego. Su uso facilita enormemente el desarrollo de videojuegos u otras aplicaciones que requieran gráficos en tiempo real.

Las partes o módulos más comunes e importantes de los que dispone un motor de juego son: el motor gráfico encargado de renderizar los gráficos 2D y 3D, el motor físico que simula los comportamientos físicos de los elementos del juego, módulo de comunicaciones, de gestión de recursos, de gestión de entrada y salida, etc. Dependiendo de la sofisticación del motor, existirán algunos que únicamente ofrezcan herramientas para los apartados gráficos y otros que mediante módulos permitirán escoger al desarrollador de cuales hará uso entre todos los ofrecidos. Es muy común que la parte relacionada con el renderizado de gráficos se haga usando a su vez librerías gráficas tales como Direct3D [17] u OpenGL [18].

La principal diferencia entre un motor y una librería gráfica es el nivel de abstracción ofrecido, ya que mientras una librería permitirá al desarrollador resolver los problemas con los que se encuentre a un nivel más bajo, los motores por su parte los resolverán sin permitir al desarrollador escoger la solución. Además los motores incluyen herramientas gráficas que harán más sencilla la labor de implementación (modelado de personajes, escenarios, físicas, sonidos, etc.).

3.6.3. ANÁLISIS DE LIBRERÍAS Y MOTORES DE JUEGOS

En este apartado se detallan algunos de los motores y librerías para el desarrollo de videojuegos que existen actualmente en el mercado:

DIRECTX (VERSIÓN 11)

Características

Conjunto de librerías de desarrollo para plataformas de Microsoft. Incluye elementos cuya finalidad es facilitar el desarrollo de videojuegos. El API está compuesto por los siguientes componentes:

- **DirectDraw:** Encargado de la representación de gráficos bidimensionales.
- **Direct3D:** Encargado de la representación de gráficos tridimensionales.
- **DirectWrite:** Encargado de la representación de las fuentes que se encuentren en el sistema operativo.
- **DirectCompute:** Encargado de gestionar el uso de la GPU y del cómputo que le será asignado.
- **DirectInput:** Encargado de detectar y gestionar los periféricos usados en la interacción del usuario.
- **DirectPlay:** Encargado de dar soporte a las comunicaciones a través de internet.
- **DirectSound:** Encargado de la reproducción y control de sonido.
- **DirectSound3D:** De forma similar a la anterior, encargado del sonido 3D.

Licencia

El desarrollo y comercialización de productos que hagan uso de esta librería disponen de una licencia gratuita.

Requisitos

- Sistema operativo de Microsoft Windows 7, Windows Vista o Windows Server 2008.
- Tarjeta gráfica compatible con DirectX 11.

Lenguaje de programación

Lenguajes basados en tecnología **.NET**.

Plataforma de ejecución

Equipos con sistemas operativos de Microsoft Windows.

IRRLICHT

Características

Motor gratuito programado en **C++** que permite el desarrollo en el mismo lenguaje (de forma no oficial es posible programar en **Java, Python**, **Perl**, etc.). Uno de sus puntos débiles es la carencia de aplicaciones para el tratamiento de los distintos medios (sonido, modelos, etc.). Entre sus principales características se encuentran:

- **Rendimiento 3D:** Mediante el uso de Direct3D y OpenGL consigue altas prestaciones en el renderizado tridimensional.
- **Soporte de shaders:** Tanto para PixelShader como VertexShader.
- **Independencia de plataforma:** Programación al margen del hardware.
- **Compatibilidad de modelos y texturas:** Soporta la carga de modelos en formatos OBJ, 3DS, B3d, BSP, MD2, etc. y de texturas en formato BMP, PNG, PSD, JPG, TGA, etc.
- **Detección de colisiones:** Entre los distintos modelos de los que hagamos uso.
- **Efectos**: El motor incluye efectos predefinidos tales transparencias, iluminación, skyboxes [19], sistemas de partículas (agua, fuego, nieve, humo, etc.), niebla, etc.
- **Sistema jerárquico de nodos:** Los elementos representados en la aplicación se conciben como nodos con un único padre y de los que dependen a su vez otros nodos, formando así un sistema jerárquico.

Licencia

El desarrollo y comercialización de productos que hagan uso de esta librería disponen de una licencia gratuita, excepto que se haga uso de la librería JPEG UG.

Requisitos

- Sistema operativo de Microsoft Windows XP o superior, Mac OSX o Linux.

Lenguaje de programación

El motor permite el desarrollo en los lenguajes C++, **C#**, Java, Perl, Ruby, Basic, Python, etc.

Plataforma de ejecución

Cualquier plataforma compatible con las librerías Direct3D y OpenGL.

XNA

Características

Se trata de una librería considerada un motor de juego de bajo nivel. Esto se debe a su nivel de abstracción, ya que ofrece un marco de desarrollo que incluye el ciclo de ejecución típico de un videojuego, aunque la mayor parte de las soluciones escogidas en el desarrollo deben ser implementadas mediante programación.

La librearía se encuentra en constante evolución y con el tiempo ha ido ampliando sus funcionalidades y comunidad. Esta última es un gran grupo de usuarios que constantemente producen tutoriales, ejemplos y contenido.

Entre las principales características de XNA podemos encontrar:

- **Representación 3D y 2D:** Ofrece funciones para el renderizado de modelos y primitivas gráficas tanto tridimensional como bidimensionalmente.
- **Multiplataforma:** Compatible con todas las plataformas de Microsoft (Microsoft Windows, Xbox 360, Zune, etc.).
- **Cargador de Contenido:** La librería incluye mecanismos para la transformación de contenidos a los formatos compatibles en tiempo de compilación.
- **Compatible con interfaces avanzadas:** Tales como dispositivos táctiles, acelerómetros, etc.
- **Funcionalidades de XBOX Live**: Debido a su compatibilidad con la plataforma, incluye funcionalidades como la reproducción de video incrustado en escenarios 3D, el sistema de invitaciones para multijugador, representación de avatares, etc.

Licencia

Comercialización y desarrollo son gratuitos con la librería XNA de Microsoft, exceptuando la publicación del videojuego en Xbox Live que requiere el pago de una cuota anual.

Requisitos

- Microsoft Visual Studio 2005 o superior.
- Tarjeta gráfica compatible con DirectX y Shader Model.

Lenguaje de programación

La implementación se realizará mediante el lenguaje de programación C#.

Plataforma de ejecución

Como ya se ha mencionado, cualquier plataforma de Microsoft (PC, Xbox 360 y Zune) es compatible con la ejecución de cualquier aplicación desarrollada con la librería.

OGRE 3D

Características

OGRE 3D es un motor gráfico de código abierto que ofrece programación orientada a objetos en C++, multiplataforma (Microsoft Windows, Mac OSX o Linux) y flexible. El motor explota el rendimiento de las tarjetas gráficas dando soporte para PixelShader y VertexShader. Debido a tratarse de un proyecto de código libre, es uno de los más extendidos y valorado por la comunicad.

Las características más relevantes de este motor son:

- **Gráficos 3D:** Aísla e independiza al usuario de la interacciones de bajo nivel realizadas con las librerías OpenGL y Direct3D.
- **Sencillez de programación:** Debido a su orientación a objetos, a las facilidades de alto nivel ofrecidas y a la extensa documentación existente.
- **Multiplataforma:** Compatible con gran variedad de plataformas (Microsoft Windows, Linux, Mac OSX).
- **Shaders de bajo y alto nivel:** Soporte para efectos y diversos lenguajes de definición de shaders (ensamblador, HLSL, DirectX, etc.).
- **Texturas en tiempo real:** El motor permite la carga de texturas para los diversos modelos en tiempo real. Soporta además múltiples formatos (PNG, JPEG, TGA, etc.).
- **Elementos predefinidos:** Desde clases de gestión de escenas comunes, hasta jerarquías de elementos mostrados, sistemas de partículas, skyboxes, etc.

Licencia

El desarrollo mediante la utilización de este motor es gratuito, aunque su comercialización se ajustará a unas condiciones en función de la versión del motor (es preciso indicar las modificaciones realizadas al motor o incluir textos definidos por el MIT).

Requisitos

- Microsoft Windows XP o superior, Linux o Mac OSX.
- Un compilador que variará en función de la plataforma.

Lenguaje de programación

En Microsoft Windows podrá hacerse uso de C++, **GCC** 3+ en Linux y **Xcode** en Mac OSX.

La implementación se realizará mediante el lenguaje de programación C#.

Plataforma de ejecución

Sistemas operativos Microsoft Windows XP o superiores, Linux y Mac OSX.

UNREAL ENGINE

Características

Uno de los más famosos motores de desarrollo de videojuegos en el mercado actual. Escrito en lenguaje C++, su repertorio de funcionalidades ha dado luz a éxitos en ventas tales como **Gears of War**, **Bioshock**, **Borderlands**, etc. Ofrece gran multitud de herramientas para la creación de contenidos de alto nivel tales como animaciones, personajes, escenarios, etc.

Las características más relevantes de este motor son:

- **Renderizado multihilo:** Renderizado tridimensional utilizando varios hilos y explotando por tanto los actuales sistemas de varios núcleos.
- **Efectos de post procesado:** Los más recientes y utilizados, tales como blur, bloom, depth of field, etc.
- **Físicas:** Soporte para la tecnología **Nvidia PhysX** que permite emular físicas altamente realistas. Incluye además deformación de materiales por VertexShader, materiales flexibles, elásticos y destrucción de estructuras del entorno.
- **Sonido 3D:** Reproducción de sonidos y música haciendo uso del posicionamiento 3D, incluyendo sonidos de ambiente.
- **Modelos:** Permite importación de modelos en los formatos 3DS, Maya, XSI, etc. Además soporta la animación de los mismos basada en esqueletos.
- **Inteligencia artificial:** Rutinas comunes de inteligencia artificial.
- **Variedad de editores:** Incluye editores de terreno, materiales, mallas, animaciones, físicas, cinemáticas, sonidos, etc.

Licencia

Aunque el desarrollo es gratuito, un uso con fines comerciales requiere el pago de una licencia que establece los porcentajes de beneficios que son propiedad de **UDK**.

Requisitos

- Microsoft Windows XP o superior.

Lenguaje de programación

Aunque se utilicen herramientas para la generación y carga de contenido, dispone de un lenguaje de scripting orientado a objetos para realizar dichas tareas.

Plataforma de ejecución

Sistemas operativos Microsoft Windows XP o superiores, Linux, Mac OSX, Xbox 360 y PlayStation 3.

4. ANÁLISIS

Este apartado recoge la fase de análisis del proyecto que servirá de base para el posterior diseño de la aplicación. Para ello se recoge una descripción general que incluye su estructura y principales funcionalidades, posteriormente será debidamente detallada con la identificación de los requisitos de usuario, con la definición de los casos de uso y con la especificación de los requisitos software que describan en detalle las características a implementar.

4.1. DESCRIPCIÓN GENERAL

La aplicación que quiere desarrollarse en este proyecto fin de carrera es un videojuego MMORTS. La aplicación se denominará Foundation haciendo referencia al libro de Isaac Asimov, el que además se tomará como fuente de inspiración para la temática del videojuego.

A grandes rasgos, Foundation es un juego de estrategia multijugador masivo en el que sus usuarios deberán competir por el control del universo. Para conseguir este cometido los jugadores deberán conquistar planetas y diversos sistemas en busca de recursos haciendo uso de sus flotas.

El juego dispone de un universo que será el escenario de juego y que además estará compuesto por varios sistemas. Estos sistemas pueden tener o no planetas y estrellas. Cada uno de estos planetas dispone de un conjunto de territorios que pueden pertenecer únicamente a un jugador. Estos territorios a su vez disponen de recursos que pueden ser extraídos mediante edificaciones. Los jugadores podrán utilizar los territorios bajo su control y los recursos extraídos para producir naves de combate con las que podrán viajar a otros sistemas y conquistar los territorios de otros jugadores.

Los jugadores podrán elegir entre varios tipos de naves en función de los recursos de los que dispongan y de las necesidades bélicas que afronten, además las naves podrán ser mejoradas mediante investigación.

4.2. DECISIÓN TECNOLÓGICA

La primera decisión tomada fue acerca del motor gráfico, optando finalmente por XNA se valoró enormemente que fuese gratuita y compatible con varios dispositivos que disponen de un amplio mercado de usuarios. Otro punto a favor fue la gran cantidad de documentación, de tutoriales y la extensa comunidad de usuarios y colaboradores que hacen uso de la librería, ya que facilita enormemente la labor de aprendizaje sobre el uso de motores de videojuegos.

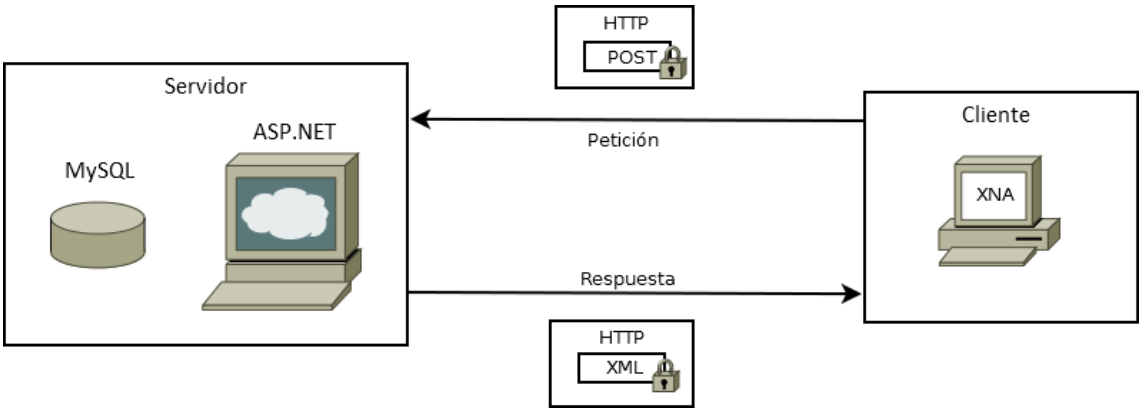
En definitiva, se trata de una herramienta potente y con mucho soporte que al estar diseñada para desarrolladores amateur, destaca por su facilidad de uso.

Intentando facilitar la tarea de implementación, se decidió escoger un protocolo de la capa de aplicación para realizar las comunicaciones entre cliente y servidor. De entre los estudiados, el que más encaja en la arquitectura cliente servidor y el que más posibilidades permite en cuanto a los tipos de datos transmitidos es HTTP. Además puede considerarse como el protocolo más usado y con el que más me encuentro familiarizado.

Respecto a la elección de XML para estructurar la información enviada del servidor al cliente, la principal razón fue la sencillez del lenguaje y la multitud de librerías disponibles para componer e interpretar documentos en este formato.

Tras la elección de XNA, la programación en C# era obligatoria, lo que condicionó mucho la elección del framework de desarrollo web, ya que ASP.NET a diferencia de J2EE, permitía programar en el mismo lenguaje. Por último mi familiaridad personal con MySQL y que se trate de una base de datos gratuita, fueron los puntos que decantaron la elección.

La siguiente ilustración, aunque no sigue ningún estándar, pretende facilitar la comprensión de la estructura cliente servidor escogida y de las tecnologías utilizadas:



5. Ilustración 19: Vista general de la aplicación

CLIENTE

Los clientes de la aplicación estarán construidos mediante la librería de desarrollo de videojuegos XNA. La implementación incluirá shaders, efectos de post procesado, sistemas de partículas, modelado procedural e interacción con el usuario. Esta última funcionalidad hará uso de los mecanismos ofrecidos por XNA y del paquete **Windows Forms**.

COMUNICACIÓN

La comunicación cliente servidor se realizará por medio de protocolo de nivel de aplicación HTTP. Las peticiones por parte del cliente harán uso del comando POST, mientras que las respuestas del servidor estarán compuestas por XML. Además la información transmitida en ambos sentidos será cifrada a nivel de aplicación mediante el algoritmo Rijndael. La clave de sesión del cifrador será intercambiada mediante el algoritmo Diffie-Hellman y la integridad de los mensajes se comprobará mediante uso de resúmenes realizados con el algoritmo **SHA-512** [20].

SERVIDOR

El servidor se implementará haciendo uso del framework ASP.NET y de los mecanismos que ofrece. El estado del juego será almacenado de forma persistente mediante una base de datos MySQL, que además se encargará de gestionar la concurrencia de datos. La concurrencia se implementará mediante procesos ligeros que gestionarán sus secciones críticas haciendo uso de elementos de control del sistema operativo.

4.3. DEFINICIÓN DE REQUISITOS DE USUARIO

El análisis de un proyecto software se materializa en la definición de un conjunto de requisitos de usuario, que especifican a alto nivel las funcionalidades o capacidades que cubrirá el sistema y las restricciones impuestas al desarrollo del mismo. Estos requisitos se recogerán en tablas que por medio

de un conjunto de campos describan sus principales características. A continuación, se muestra la plantilla de tabla utilizada para definir los requisitos de usuario:

Identificador:			
Nombre			
Descripción			
Prioridad		**Necesidad**	
Estabilidad		**Verificabilidad**	

Ilustración 20: Plantilla de requisito de usuario

El significado de los campos de la tabla es el siguiente:

- **Identificador:** código alfanumérico que identifica de forma unívoca cada requisito de usuario. La sintaxis del identificador se define de la siguiente forma:

  RU<tipo>-<número>

  - *RU:* hace referencia a que el requisito es de tipo "requisito de usuario".
  - *<Tipo>:* es un carácter que identifica el tipo de requisito de usuario, "C" para los requisitos de capacidad o funcionalidad de la aplicación, y "R" para los requisitos de restricción.
  - *<Número>:* es una cifra de dos dígitos que identifica al requisito dentro de los requisitos de su mismo tipo. Empieza en 01 y crece una unidad por cada requisito.

  Ejemplos: RUC-08, RUC-10, RUR-02, RUR-11, etc.

- **Nombre:** descripción en pocas palabras de la funcionalidad de un requisito, más fácil de recordar que el identificador y más breve que la descripción formal del mismo.
- **Descripción:** descripción textual breve del requisito, detallando su información de forma clara y concisa.
- **Prioridad:** nivel de preferencia de la implementación del requisito durante la fase de desarrollo de la aplicación. Puede tomar los valores "Alta", "Media" o "Baja".
- **Necesidad:** grado en que se puede prescindir o no de la implementación de un requisito. Puede tomar los valores "Esencial", "Deseable" u "Opcional".
- **Estabilidad:** indica si el requisito puede sufrir o no cambios de importancia durante el desarrollo del proyecto. Puede tomar los valores "Estable" o "No estable".
- **Verificabilidad:** indica el grado en que se puede comprobar que el requisito ha sido implementado en la aplicación. Puede tomar los valores "Alta", "Media" o "Baja".

4.3.1. REQUISITOS DE CAPACIDAD

Identificador: RUC-01			
Nombre	Identificación de jugador		
Descripción	Los jugadores registrados podrán acceder a la aplicación introduciendo su nombre de usuario y contraseña.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 1: Requisito de capacidad RUC-01

Identificador: RUC-02			
Nombre	Registro de jugadores		
Descripción	Los jugadores podrán registrarse en la aplicación indicando: • **Nombre de usuario:** Nombre de jugador usado para acceder a la aplicación. • **Email:** Dirección de correo bien formada del jugador. • **Contraseña:** Clave secreta que será necesario introducir dos veces para acceder a la aplicación. • **Color Primario:** Primer color que identificará al jugador. • **Color Secundario:** Segundo color que identificará al jugador. • **Bandera:** Imagen que utilizará el jugador para identificarse visualmente.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	No Estable	**Verificabilidad**	Alta

Tabla 2: Requisito de capacidad RUC-02

Identificador: RUC-03			
Nombre	Mensajes de error		
Descripción	Siempre que se produzca un error en la aplicación o el jugador intente realizar una acción prohibida, deberá mostrarse un mensaje informando acerca de ello.		
Prioridad	Media	**Necesidad**	Deseable
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 3: Requisito de capacidad RUC-03

Identificador: RUC-04			
Nombre	Mensajes de ayuda		
Descripción	La aplicación deberá mostrar mensajes de ayuda para facilitar la comprensión de las acciones más complejas conceptualmente.		
Prioridad	Media	**Necesidad**	Deseable
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 4: Requisito de capacidad RUC-04

Identificador: RUC-05			
Nombre	Información de jugador		
Descripción	Una vez el jugador haya sido identificado, se mostrará información acerca de sus recursos, de su nivel de mejoras y de su latencia con el servidor.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 5: Requisito de capacidad RUC-05

Identificador: RUC-06			
Nombre	Información de elementos del escenario		
Descripción	Deberá mostrarse información acerca de los elementos del escenario con los que el jugador interactúe.		
Prioridad	Media	**Necesidad**	Deseable
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 6: Requisito de capacidad RUC-06

Identificador: RUC-07			
Nombre	Acciones sobre sistemas		
Descripción	El jugador podrá acceder a los sistemas en los que tenga naves de batalla, territorios o lunas para interactuar con ellos.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 7: Requisito de capacidad RUC-07

Identificador: RUC-08			
Nombre	Acciones sobre planetas		
Descripción	El jugador podrá acceder a los planetas para interactuar con ellos.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 8: Requisito de capacidad RUC-08

Identificador: RUC-09			
Nombre	Construcción		

Descripción	Los jugadores podrán construir sobre territorios o lunas que estén bajo su control. Las edificaciones que pueden construir son:
	• **Territorio:**
	  ○ **Ciudad:** Permite construir naves de batalla.
	  ○ **Laboratorio:** Permite realizar mejoras sobre las naves de batalla.
	  ○ **Extractor:** Permite obtener éter del territorio.
	  ○ **Mina:** Permite obtener hefestos del territorio.
	  ○ **Defensas:** Atacan de forma automática a los jugadores que intenten conquistar el territorio.
	• **Luna:**
	  ○ **Base lunar:** Permite construir naves de batalla.		

Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 9: Requisito de capacidad RUC-09

Identificador: RUC-10			
Nombre	Acciones sobre territorios con ciudad		
Descripción	Los territorios que dispongan de una ciudad construida podrán crear naves de batalla que aparecerán en la órbita del planeta. Las naves de batalla que es posible crear son: • **Pegasus:** Nave de batalla ligera. • **Unicorn:** Nave de batalla media capaz de geoformar planetas. • **Chimera:** Nave de batalla pesada.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 10: Requisito de capacidad RUC-10

Identificador: RUC-11	
Nombre	Recarga de naves de batalla
Descripción	Las naves de batalla recargarán el número de ataques y movimientos

	de forma periódica.		
Prioridad	Media	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 11: Requisito de capacidad RUC-11

Identificador: RUC-12			
Nombre	Acciones sobre naves de batalla		

Descripción	Las acciones que el jugador puede realizar sobre sus naves de batalla son:
	• **Mover:** Desplaza la nave de batalla a un cuadrado adyacente.
	• **Atacar:** Ataca a naves de batalla que se encuentren en posiciones adyacentes.
	• **Saltar:** Utilizado para desplazarse a sistemas adyacentes cuando la nave de batalla se encuentra en los límites de su propio sistema.
	• **Conquistar:** Utilizado para conquistar territorios de planetas que se encuentren en una posición adyacente.
	• **Cambiar modo:**
	  ○ **Defensivo:** La unidad atacará en caso de recibir ataques.
	  ○ **Ofensivo:** La unidad no atacará en caso de recibir ataques.
	• **Geoformar:** Acción únicamente permitida para la nave de batalla unicorn, permite geoformar un planeta adyacente.		

Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 12: Requisito de capacidad RUC-12

Identificador: RUC-13			
Nombre	Acciones sobre lunas con base lunar		
Descripción	Las lunas que dispongan de una base lunar construida podrán crear naves de batalla que aparecerán en la órbita del planeta. Las naves de batalla que es posible crear son: • **Pegasus:** Nave de batalla ligera.		
Prioridad	Media	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 13: Requisito de capacidad RUC-13

Identificador: RUC-14			
Nombre	Acciones sobre territorios con laboratorio		
Descripción	Los territorios con laboratorio permitirán realizar las siguientes mejoras de las naves de batalla: • **Ataque:** Incrementa el ataque de las naves de batalla. • **Defensa:** incrementa la defensa de las naves de batalla.		
Prioridad	Media	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 14: Requisito de capacidad RUC-14

Identificador: RUC-15			
Nombre	Recolección de recursos		
Descripción	Los territorios que tengan un extractor o una mina harán incrementar los recursos del jugador de forma periódica. Las minas incrementarán el hefestos del jugador y los extractores incrementarán el éter. La población será fija y dependerá únicamente del número de territorios que estén bajo el control del jugador.		
Prioridad	Media	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 15: Requisito de capacidad RUC-15

Identificador: RUC-16			
Nombre	Generación dinámica del universo		
Descripción	El universo de la aplicación deberá generarse dinámicamente y crecer según sea necesario por el incremento de jugadores. Los sistemas, planetas, territorios y lunas deberán generarse con atributos y características visuales distintas. Además la distribución de tales elementos deberá también ser generada y evitar colisiones.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Media

Tabla 16: Requisito de capacidad RUC-16

Identificador: RUC-17			
Nombre	Acciones de jugadores		
Descripción	Los jugadores deberán visualizar la secuencia de acciones que realizan el resto de jugadores.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Media

Tabla 17: Requisito de capacidad RUC-17

Identificador: RUC-18			
Nombre	Fin del juego		
Descripción	Los jugadores que no dispongan de territorios en el universo serán eliminados del juego y no podrán acceder al mismo.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Media

Tabla 18: Requisito de capacidad RUC-18

4.3.2. REQUISITOS DE RESTRICCIÓN

Identificador: RUR-01	
Nombre	Recursos de jugador
Descripción	Existen tres recursos de los que el jugador hará uso para construir edificaciones, actualizar mejoras o crear naves de batalla: • **Hefestos:** Mineral obtenido de los territorios que disponen de una mina. • **Éter:** Gas obtenido de los territorios que disponen de un

	extractor. • **Población:** Cantidad de naves de batalla que el jugador puede crear y obtenido de los territorios que están bajo el control del jugador.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 19: Requisito de restricción RUR-01

Identificador: RUR-02			
Nombre	División del escenario		
Descripción	Los escenarios estarán divididos en cuadrados que formarán una cuadrícula para definir las unidades de espacio que pueden ocupar los diversos elementos del escenario.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 20: Requisito de restricción RUR-02

Identificador: RUR-03			
Nombre	Composición del universo		
Descripción	El universo estará compuesto por sistemas y por agujeros negros.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 21: Requisito de restricción RUR-03

Identificador: RUR-04			
Nombre	Composición de los sistemas		
Descripción	Los sistemas estarán compuestos por campos de meteoritos y algunos de ellos por una estrella y planetas.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 22: Requisito de restricción RUR-04

Identificador: RUR-05			
Nombre	Composición de los planetas		
Descripción	Los planetas podrán estar o no geoformados. En caso de estar geoformados sus territorios podrán ser ocupados por los diversos jugadores de la aplicación. Además de territorios, los planetas pueden disponer de lunas que también podrán ser conquistadas por los jugadores. El número de lunas y territorios varía en función del tamaño del planeta.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 23: Requisito de restricción RUR-05

Identificador: RUR-06			
Nombre	Composición de los territorios		
Descripción	Los territorios dispondrán de una cantidad de éter, de hefestos y de población.		
Prioridad	Media	**Necesidad**	Esencial

Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 24: Requisito de restricción RUR-06

Identificador: RUR-07			
Nombre	Interacción con elementos del escenario		
Descripción	Los jugadores interactuarán con los elementos del escenario pulsando sobre ellos con el botón izquierdo del ratón y pulsando sobre los botones de la interfaz para desempeñar las diversas acciones permitidas.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 25: Requisito de restricción RUR-07

Identificador: RUR-08			
Nombre	Acercar/Alejar vista		
Descripción	Los jugadores podrán acercar y alejar la vista a los elementos del escenario haciendo uso de la rueda del ratón.		
Prioridad	Media	**Necesidad**	Deseable
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 26: Requisito de restricción RUR-08

Identificador: RUR-09			
Nombre	Movimiento por el escenario		
Descripción	Los jugadores podrán moverse por el escenario haciendo uso de las flechas de dirección del teclado.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 27: Requisito de restricción RUR-09

Identificador: RUR-10			
Nombre	Girar vista		
Descripción	Los jugadores podrán girar la vista del escenario haciendo uso de las teclas Repág y Avpág.		
Prioridad	Alta	**Necesidad**	Deseable
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 28: Requisito de restricción RUR-10

Identificador: RUR-11			
Nombre	Atributos de naves de batalla		
Descripción	Las naves de batalla dispondrán de los siguientes atributos: • **Número:** Cantidad de naves de batalla del mismo tipo que conforman la agrupación • **Movimientos:** Cantidad de movimientos que puede realizar la nave de batalla. • **Ataques:** Cantidad de ataques que puede realizar la nave de batalla. • **Modo:** Puede ser ofensivo o defensivo.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 29: Requisito de restricción RUR-11

Identificador: RUR-12			
Nombre	Relación de precios		

Descripción	La construcción de edificaciones y la creación de naves de batalla requieren de los recursos del jugador. A continuación se expone la relación de precios de ambos elementos:
	• **Edificaciones:**
	  ○ **Territorio:**
	  ▪ **Ciudad:** Nivel medio de éter y alto de hefestos.
	  ▪ **Laboratorio:** Nivel medio de hefestos y alto de éter.
	  ▪ **Extractor:** Nivel bajo de hefestos.
	  ▪ **Mina:** Nivel bajo de éter.
	  ▪ **Defensa:** Nivel muy bajo de éter y hefestos.
	  ○ **Luna:**
	  ▪ **Base lunar:** Nivel alto de éter y hefestos.
	• **Naves de batalla:**
	  ○ **Pegasus:** Bajo nivel de éter, hefestos y población.
	  ○ **Unicorn:** Medio nivel de éter, hefestos y población.
	  ○ **Chimera:** Alto nivel de éter, hefestos y población		

Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 30: Requisito de restricción RUR-12

Identificador: RUR-13			
Nombre	Visibilidad		
Descripción	Los jugadores únicamente podrán visualizar aquellos sistemas que dispongan de naves de batalla, territorios o lunas bajo su control y los adyacentes a los descritos.		
Prioridad	Media	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Media

Tabla 31: Requisito de restricción RUR-13

Identificador: RUR-14			
Nombre	Realismo		
Descripción	La aplicación deberá generar los elementos del escenario con cierta semejanza al universo conocido.		
Prioridad	Media	**Necesidad**	Opcional
Estabilidad	No Estable	**Verificabilidad**	Baja

Tabla 32: Requisito de restricción RUR-14

Identificador: RUR-15			
Nombre	Comunicaciones seguras		
Descripción	Las comunicaciones entre cliente y servidor deben ser seguras.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Media

Tabla 33: Requisito de restricción RUR-15

Identificador: RUR-16			
Nombre	Estado de juego		
Descripción	El servidor se encargará de mantener el estado del juego y de todos sus elementos de forma consistente y persistente.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Media

Tabla 34: Requisito de restricción RUR-16

Identificador: RUR-17			
Nombre	Rendimiento		
Descripción	Se utilizarán técnicas de optimización del renderizado para mejorar el rendimiento de la aplicación.		
Prioridad	Baja	**Necesidad**	Opcional
Estabilidad	Estable	**Verificabilidad**	Baja

Tabla 35: Requisito de restricción RUR-17

Identificador: RUR-18			
Nombre	Tolerancia a fallos		
Descripción	El servidor deberá ser tolerante a errores y por tanto ser capaz de recuperarse de cualquier error o imprevisto. Los errores que surjan se almacenarán para su posterior estudio.		
Prioridad	Media	**Necesidad**	Deseable
Estabilidad	Estable	**Verificabilidad**	Media

Tabla 36: Requisito de restricción RUR-18

Identificador: RUR-19			
Nombre	Inactividad de jugadores		
Descripción	El juego debe contemplar la inactividad de los jugadores debido a que el tiempo de juego no se detiene.		
Prioridad	Media	**Necesidad**	Deseable
Estabilidad	Estable	**Verificabilidad**	Media

Tabla 37: Requisito de restricción RUR-19

4.4. DEFINICIÓN DE CASOS DE USO

En este apartado van a modelarse y definirse textualmente los casos de uso de la aplicación, que describen las funcionalidades principales de la misma y la interacción de los usuarios en diferentes escenarios. Se han distinguido los siguientes actores analizando tanto cliente como servidor:

- **Cliente:**
  - **Jugador:** El jugador y principal usuario de la aplicación cliente.
  - **Tiempo Cliente:** Acciones llevadas a cabo de forma periódica en el cliente.
- **Servidor:**
  - **Tiempo Servidor:** Acciones llevadas a cabo de forma periódica en el servidor.

A continuación se muestran los diagramas de casos de uso relativos a los tres roles:

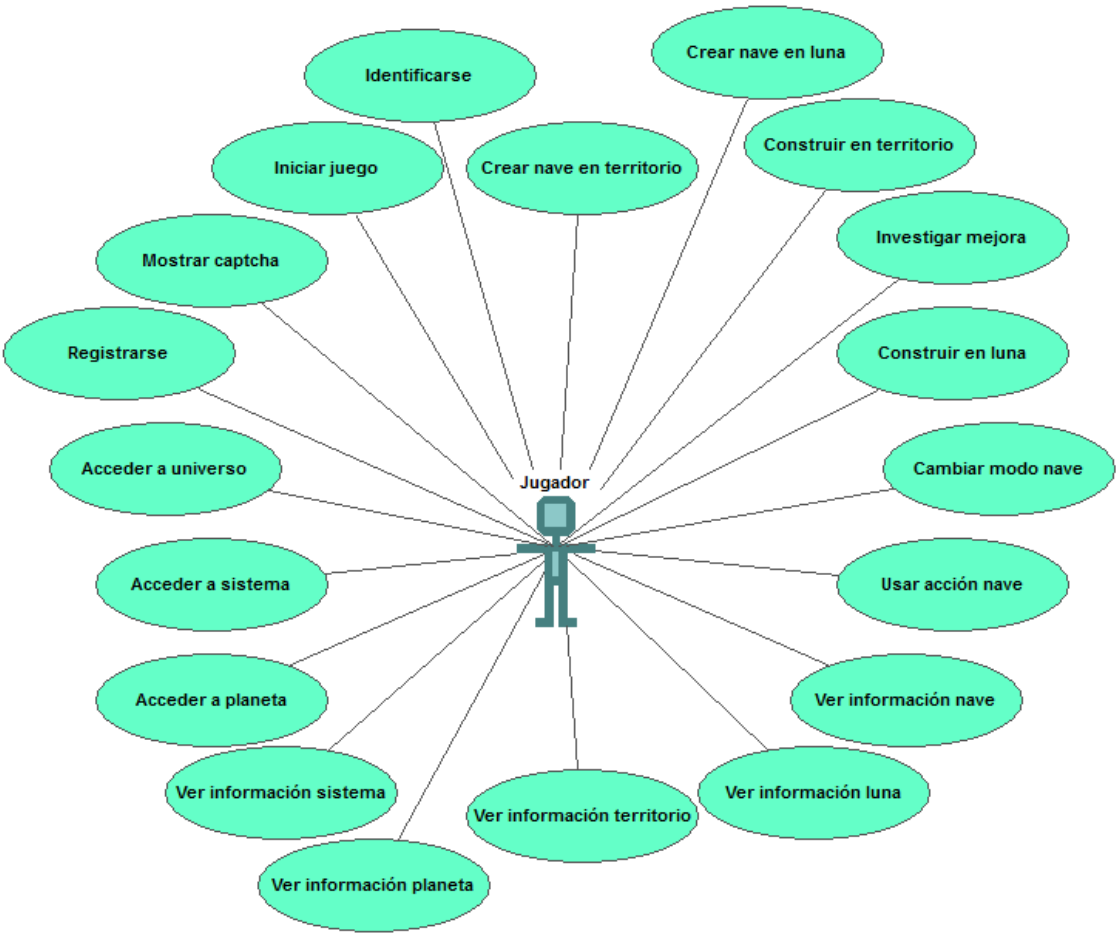


Ilustración 21: Diagrama de caso de uso del rol Jugador

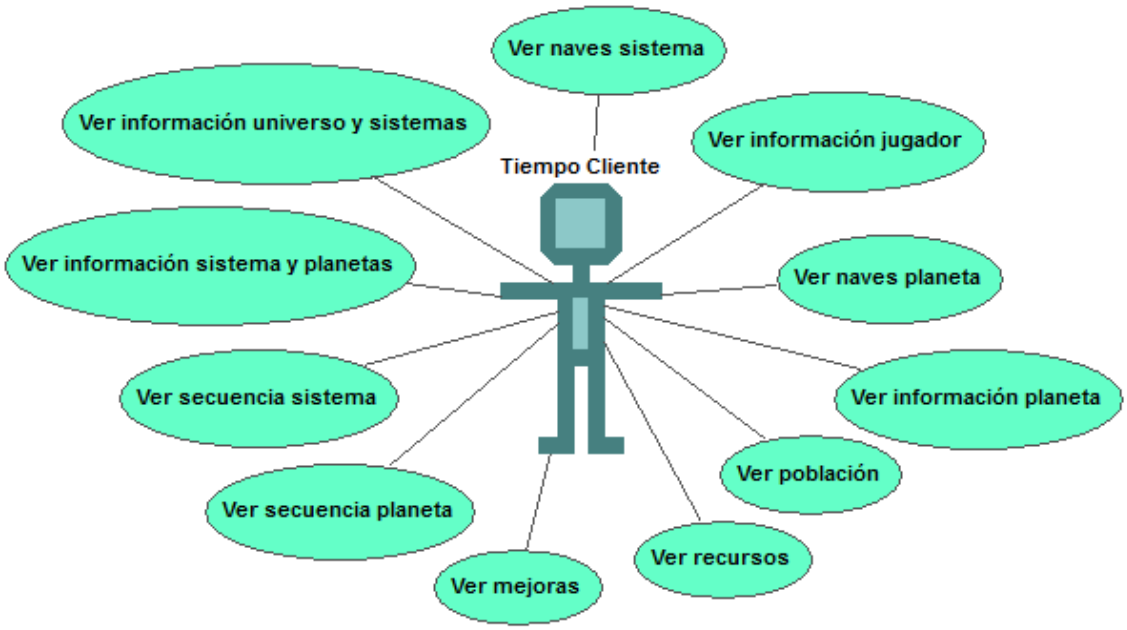


Ilustración 22: Diagrama de caso de uso del rol Tiempo Cliente

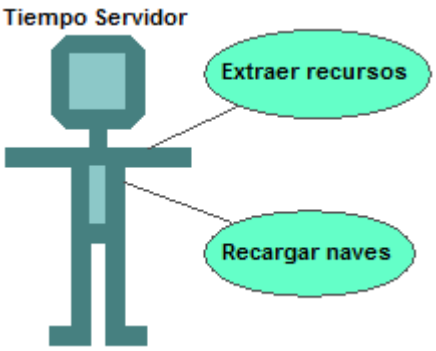


Ilustración 23: Diagrama de caso de uso del rol Tiempo Servidor

A continuación, se van a describir textualmente cada uno de los casos de uso de la aplicación. Para ello, se usará una plantilla de tabla conteniendo los siguientes campos:

Identificador:	
Nombre	
Objetivo	
Precondiciones	
Postcondiciones	
Escenario básico	
Escenario alternativo	

Tabla 38: Plantilla de descripción de casos de uso

El significado de los campos de la tabla es el siguiente:

- **Identificador:** código alfanumérico que identifica de forma unívoca cada caso de uso. La sintaxis del identificador se define de la siguiente forma:

  CU-<número>

  - *CU:* abreviatura de "caso de uso".
  - *<Número>:* es una cifra de dos dígitos que identifica al caso de uso. Empieza en 01 y crece una unidad por cada requisito.

  Ejemplos: CU-02, CU-11, etc.

- **Nombre:** nombre del caso de uso, que resume la funcionalidad referida por el mismo.
- **Actor:** nombre del actor que inicia el caso de uso.
- **Objetivo:** breve descripción de la funcionalidad del caso de uso.
- **Precondiciones:** lista de todas las condiciones que deben cumplirse y acciones que deben haber tenido lugar, de forma previa al inicio del caso de uso.
- **Postcondiciones:** efectos que provoca la ejecución del caso de uso.
- **Escenario básico:** enumeración ordenada de las acciones del usuario y respuestas del sistema, que tendrá lugar durante la ejecución del caso de uso en condiciones normales. Dicho de otro modo, son los pasos que se ha de seguir para ejecutar la funcionalidad del caso de uso.
- **Escenario alternativo:** bifurcación en la secuencia de pasos descrita por el escenario básico, que describe los casos en los que la funcionalidad esperada no puede llevarse a cabo por unos motivos determinados.

4.4.1. DESCRIPCIÓN DE LOS CASOS DE USO

Identificador: CU-01	
Nombre	Iniciar juego
Actor	Jugador
Objetivo	Ejecutar la aplicación y realizar la negociación de la clave se sesión.
Precondiciones	El servidor está en línea, la aplicación se está ejecutando y la base de datos está operativa.
Postcondiciones	El cliente y el servidor comparten la clave de sesión.
Escenario básico	1. Se ejecutar la aplicación. 2. Se comprueba que el servidor está en línea. 3. Se realiza el intercambio de claves. 4. Se muestra un mensaje indicando que el servidor está en línea.
Escenario alternativo	4. Se muestra un mensaje indicando que se ha producido un error en la conexión.
	3. Se muestra un mensaje indicando que se ha producido un error de seguridad.

Tabla 39: Caso de uso CU-01

Identificador: CU-02	
Nombre	Registrarse
Actor	Jugador
Objetivo	Enviar los datos de jugador al servidor para que pueda almacenarlos y tener así acceso a la aplicación.
Precondiciones	El servidor está en línea, con la aplicación en ejecución y la base de datos está operativa.
Postcondiciones	El jugador registra sus datos en la aplicación.
Escenario básico	1. Se ejecutar la aplicación. 2. Se accede al menú de registro. 3. Se introducen los datos del jugador. 4. Se introduce la contraseña del jugador dos veces. 5. Se seleccionan los colores primario y secundario. 6. Se selecciona una foto que será usada como bandera. 7. Se introduce el texto asociado al **Captcha**. 8. Se acepta el formulario pulsando en el botón de registro. 9. Se comprueba que los datos introducidos son correctos. 10. Se muestra un mensaje que indica que el registro se ha realizado satisfactoriamente.
Escenario alternativo	9. Se muestra un mensaje de error indicando que se ha producido un error en la conexión.
	9. Se muestra un mensaje de error indicando que se ha producido un error de seguridad.
	9. Se muestra un mensaje de error indicando que el nombre de jugador, el email o los colores ya están registrados.
	9. Se muestra un mensaje de error indicando que las contraseñas no coinciden.
	9. Se muestra un mensaje de error indicando que el email es incorrecto.
	9. Se muestra un mensaje de error indicando que el nombre de jugador es demasiado corto.
	9. Se muestra un mensaje de error indicando que el color primario y secundario coinciden.
	9. Se muestra un mensaje de error indicando que el texto asociado al captcha no es correcto.

Tabla 40: Caso de uso CU-02

Identificador: CU-03	
Nombre	Mostrar captcha
Actor	Jugador
Objetivo	Mostrar un captcha al jugador.
Precondiciones	
Postcondiciones	La aplicación muestra el captcha al jugador.
Escenario básico	1. Se ejecuta la aplicación. 2. Se accede al menú de registro. 3. Se muestra el captcha al jugador.
Escenario alternativo	3. Se muestra una imagen negra al jugador.

Tabla 41: Caso de uso CU-03

Identificador: CU-04	
Nombre	Acceder a universo
Actor	Jugador
Objetivo	Mostrar el universo al jugador.
Precondiciones	El jugador está identificado en la aplicación.
Postcondiciones	La aplicación muestra el universo al jugador.
Escenario básico	1. Se identifica el jugador en la aplicación. 2. Se muestra el universo al jugador.
Escenario alternativo	2. Se muestra un mensaje al jugador indicando que ha sido derrotado.

Tabla 42: Caso de uso CU-04

Identificador: CU-05	
Nombre	Ver información sistema
Actor	Jugador
Objetivo	Mostrar información de un sistema al jugador acerca de sus planetas y de los jugadores que poseen territorios en estos últimos.
Precondiciones	El jugador está identificado en la aplicación.
Postcondiciones	La aplicación muestra información sobre sus recursos, mejoras y latencia al jugador.
Escenario básico	1. Se identifica el jugador en la aplicación. 2. Se selecciona un sistema del universo. 3. Se muestra la información al jugador.
Escenario alternativo	

Tabla 43: Caso de uso CU-05

Identificador: CU-06	
Nombre	Acceder a sistema
Actor	Jugador
Objetivo	Mostrar al jugador el sistema.
Precondiciones	Territorios o naves de batalla del sistema están bajo el control del jugador.
Postcondiciones	La aplicación muestra el sistema al jugador con sus campos de meteoritos, las naves de combate de los jugadores y en caso de tenerlos, su estrella y planetas.
Escenario básico	1. Se identifica el jugador en la aplicación. 2. Se selecciona un sistema del universo. 3. Se comprueba que existen territorios o naves de batalla en el sistema bajo el control del jugador. 4. Se muestra el sistema al jugador.

Escenario alternativo	3. Se muestra un mensaje al jugador indicando el sistema no es visible.

Tabla 44: Caso de uso CU-06

Identificador: CU-07	
Nombre	Acceder a sistema
Actor	Jugador
Objetivo	Mostrar al jugador el sistema.
Precondiciones	Territorios o naves de batalla del sistema están bajo el control del jugador.
Postcondiciones	La aplicación muestra el sistema al jugador con sus campos de meteoritos, las naves de combate de los jugadores y en caso de tenerlos, su estrella y planetas.
Escenario básico	1. Se identifica el jugador en la aplicación. 2. Se selecciona un sistema del universo. 3. Se comprueba que existen territorios o naves de batalla en el sistema bajo el control del jugador. 4. Se muestra el sistema al jugador.
Escenario alternativo	4. Se muestra un mensaje al jugador indicando el sistema no es visible.

Tabla 45: Caso de uso CU-07

Identificador: CU-08	
Nombre	Ver información planeta
Objetivo	Mostrar información de un determinado planeta.
Precondiciones	Le está siendo mostrado un sistema al jugador que dispone de planetas.
Postcondiciones	La aplicación muestra información sobre los territorios, lunas y edificaciones del planeta.
Escenario básico	1. Se accede a un sistema. 2. Se selecciona un planeta. 3. Se muestra la información al jugador.
Escenario alternativo	

Tabla 46: Caso de uso CU-08

Identificador: CU-09	
Nombre	Acceder a planeta
Actor	Jugador
Objetivo	Mostrar al jugador el planeta
Precondiciones	Le está siendo mostrado un sistema al jugador que dispone de planetas .
Postcondiciones	La aplicación muestra el planeta al jugador con sus territorios, lunas y naves de batalla en órbita.
Escenario básico	1. Se accede a un sistema. 2. Se selecciona un planeta. 3. Se accede al planeta. 4. Se muestra el planeta al jugador.
Escenario alternativo	

Tabla 47: Caso de uso CU-09

Identificador: CU-10	
Nombre	Ver información territorio
Actor	Jugador
Objetivo	Mostrar información de un determinado territorio.
Precondiciones	Le está siendo mostrado un planeta al jugador.
Postcondiciones	La aplicación muestra información sobre las edificaciones, recursos del

	territorio y del jugador que lo controla.
Escenario básico	1. Se accede a un planeta. 2. Se selecciona un territorio. 3. Se muestra la información al jugador.
Escenario alternativo	

Tabla 48: Caso de uso CU-10

Identificador: CU-11	
Nombre	Ver información luna
Actor	Jugador
Objetivo	Mostrar información de una determinada luna.
Precondiciones	Le está siendo mostrado un planeta al jugador.
Postcondiciones	La aplicación muestra información sobre las edificaciones de la luna y del jugador que lo controla.
Escenario básico	1. Se accede a un planeta. 2. Se selecciona una luna. 3. Se muestra la información al jugador.
Escenario alternativo	

Tabla 49: Caso de uso CU-11

Identificador: CU-12	
Nombre	Construir en territorio
Actor	Jugador
Objetivo	Construir una determinada edificación en un territorio.
Precondiciones	Le está siendo mostrado un planeta al jugador, el territorio está bajo el control del jugador, no está edificado el territorio y el jugador dispone de los recursos necesarios.
Postcondiciones	La aplicación construye una edificación en el territorio seleccionado por el jugador.
Escenario básico	1. Se accede a un planeta. 2. Se selecciona un territorio. 3. Se indica la edificación que se quiere construir. 4. Se comprueba que el planeta esté geoformado. 5. Se comprueba que el territorio no tenga actualmente edificaciones construidas. 6. Se comprueba que el territorio está bajo el control del jugador. 7. Se comprueba que el jugador dispone de los recursos necesarios. 8. Se construye la edificación en el territorio.
Escenario alternativo	4. Se muestra un mensaje al jugador indicando que el planeta no está geoformado.
	5. Se muestra un mensaje al jugador indicando que en el territorio ya existe una edificación.
	6. Se muestra un mensaje al jugador indicando que el territorio no está bajo el control del jugador.
	7. Se muestra un mensaje al jugador indicando que no dispone de suficientes recursos.

Tabla 50: Caso de uso CU-12

Identificador: CU-13	
Nombre	Construir en luna
Actor	Jugador
Objetivo	Construir una determinada edificación en una luna.

Precondiciones	Le está siendo mostrado un planeta al jugador, la luna está bajo el control del jugador, no está edificada la luna y el jugador dispone de los recursos necesarios.
Postcondiciones	La aplicación construye una edificación en la luna seleccionado por el jugador.
Escenario básico	1. Se accede a un planeta. 2. Se selecciona una luna. 3. Se indica la edificación que se quiere construir. 4. Se comprueba que la luna no tenga actualmente edificaciones construidas. 5. Se comprueba que la luna esté bajo el control del jugador. 6. Se comprueba que el jugador dispone de los recursos necesarios. 7. Se construye la edificación en la luna.
Escenario alternativo	4. Se muestra un mensaje al jugador indicando que en la luna ya existe una edificación. 5. Se muestra un mensaje al jugador indicando que la luna no está bajo el control del jugador. 6. Se muestra un mensaje al jugador indicando que no dispone de suficientes recursos.

Tabla 51: Caso de uso CU-13

Identificador: CU-14	
Nombre	Crear nave en territorio
Actor	Jugador
Objetivo	Crear una determinada nave de batalla en un territorio.
Precondiciones	Le está siendo mostrado un planeta al jugador, el territorio está bajo el control del jugador, el territorio dispone de una ciudad construida y el jugador dispone de los recursos necesarios.
Postcondiciones	La aplicación crea una nave de batalla en la órbita del planeta.
Escenario básico	1. Se accede a un planeta. 2. Se selecciona un territorio. 3. Se indica que nave de batalla se quiere crear. 4. Se comprueba que el territorio esté bajo el control del jugador. 5. Se comprueba que haya una ciudad construida en el territorio. 6. Se comprueba que haya espacio libre en la órbita del planeta. 7. Se comprueba que el jugador dispone de los recursos necesarios. 8. Se crea la nave de batalla.
Escenario alternativo	4. Se muestra un mensaje al jugador indicando que el territorio no está bajo el control del jugador. 5. Se muestra un mensaje al jugador indicando que no existe una ciudad construida en ese territorio. 6. Se muestra un mensaje al jugador indicando que no existe espacio libre en la órbita del planeta. 7. Se muestra un mensaje al jugador indicando que no dispone de suficientes recursos.

Tabla 52: Caso de uso CU-14

Identificador: CU-15	
Nombre	Crear nave en luna
Actor	Jugador
Objetivo	Crear una determinada nave de batalla en una luna.
Precondiciones	Le está siendo mostrado un planeta al jugador, la luna está bajo el control del jugador, la luna dispone de una base lunar construida y el jugador

	dispone de los recursos necesarios.
Postcondiciones	La aplicación crea una nave de batalla en la órbita de la luna.
Escenario básico	1. Se accede a un planeta. 2. Se selecciona una luna. 3. Se indica que se quiere crear una nave de batalla. 4. Se comprueba que la luna esté bajo el control del jugador. 5. Se comprueba que haya una base lunar construida en la luna. 6. Se comprueba que haya espacio libre en la órbita de la luna. 7. Se comprueba que el jugador dispone de los recursos necesarios. 8. Se crea la nave de batalla.
Escenario alternativo	4. Se muestra un mensaje al jugador indicando que la luna no está bajo el control del jugador.
	5. Se muestra un mensaje al jugador indicando que no existe una base lunar construida en esa luna.
	6. Se muestra un mensaje al jugador indicando que no existe espacio libre en la órbita de la luna.
	7. Se muestra un mensaje al jugador indicando que no dispone de suficientes recursos.

Tabla 53: Caso de uso CU-15

Identificador: CU-16	
Nombre	Ver información nave
Actor	Jugador
Objetivo	Mostrar información de una determinada nave de batalla.
Precondiciones	Le está siendo mostrado un sistema al jugador y el jugador dispone de naves de batalla en ese sistema.
Postcondiciones	La aplicación muestra información sobre la cantidad, movimientos, ataques y mejoras de una nave de batalla.
Escenario básico	1. Se accede a un sistema. 2. Se selecciona una nave de batalla. 3. Se muestra la información al jugador.
Escenario alternativo	

Tabla 54: Caso de uso CU-16

Identificador: CU-17	
Nombre	Usar acción nave
Actor	Jugador
Objetivo	Mover, atacar, geoformar, saltar o conquistar con una determinada nave de batalla.
Precondiciones	Le está siendo mostrado un sistema al jugador y el jugador dispone de naves de batalla en ese sistema.
Postcondiciones	La aplicación hace moverse, atacar, geoformar, saltar o conquistar a la nave de batalla seleccionada.
Escenario básico	1. Se accede a un sistema. 2. Se selecciona una nave de batalla. 3. Se indica la acción que se quiere realizar con la nave de batalla. 4. Se comprueba que la nave de batalla pertenece al jugador. 5. Según el tipo de acción se comprueba: a. En caso de ser ataque se comprueba que la nave de batalla dispone de suficientes ataques y se realice sobre otra nave de batalla. b. En caso de ser movimiento se comprueba que la nave de batalla dispone de suficientes movimientos y que se realice

	a un espacio libre. c. En caso de ser salto se comprueba que se realice a un espacio libre. d. En caso de ser conquista se comprueba que el territorio o luna no pertenece al jugador. e. En caso de ser geoformación se comprueba que la nave de batalla es del tipo unicorn y que el planeta no esté formado. 6. Se realiza la acción. 7. En caso de ser ataque y de que la nave de batalla esté en modo ofensivo, la nave atacada devuelve el ataque a la del jugador. 8. En caso de ser conquista y el territorio tener defensas, el territorio ataca a la nave de batalla.
Escenario alternativo	4. Se muestra un mensaje al jugador indicando que la nave de batalla no pertenece al jugador.
	5. Se muestra un mensaje al jugador indicando que la nave de batalla no dispone de ataques suficientes.
	5. Se muestra un mensaje al jugador indicando que no hay una nave de batalla en el espacio seleccionado.
	5. Se muestra un mensaje al jugador indicando que la nave de batalla no dispone de movimientos suficientes.
	5. Se muestra un mensaje al jugador indicando que el espacio seleccionado para moverse no está libre.
	5. Se muestra un mensaje al jugador indicando que el espacio seleccionado para saltar no está libre.
	5. Se muestra un mensaje al jugador indicando el territorio o luna no pertenece al jugador.
	5. Se muestra un mensaje al jugador indicando la nave de batalla no puede realizar geoformaciones.
	5. Se muestra un mensaje al jugador indicando que el planeta ya está geoformado.

Tabla 55: Caso de uso CU-17

Identificador: CU-18	
Nombre	Cambiar modo nave
Actor	Jugador
Objetivo	Cambiar el modo de la nave de batalla a defensivo u ofensivo.
Precondiciones	Le está siendo mostrado un sistema al jugador.
Postcondiciones	La aplicación cambia el modo de la nave de batalla.
Escenario básico	1. Se accede a un sistema. 2. Se selecciona una nave de batalla. 3. Se indica el cambio de modo.
Escenario alternativo	

Tabla 56: Caso de uso CU-18

Identificador: CU-19	
Nombre	Investigar mejora
Actor	Jugador
Objetivo	Incrementar el ataque o defensa de las naves de batalla.
Precondiciones	Le está siendo mostrado un planeta al jugador, el territorio está bajo el control del jugador, el territorio dispone de un laboratorio construido y el jugador dispone de los recursos necesarios.
Postcondiciones	La aplicación incrementa el ataque o defensa de las naves de batalla.

Escenario básico	1. Se accede a un planeta. 2. Se selecciona un territorio. 3. Se comprueba que el territorio tiene un laboratorio construido. 4. Se comprueba que el territorio está bajo el control del jugador. 5. Se comprueba que el jugador dispone de los recursos necesarios. 6. Se indica la mejora a realizar. 7. La mejora se realiza.
Escenario alternativo	3. Se muestra un mensaje al jugador indicando que el territorio no tiene un laboratorio construido.
	4. Se muestra un mensaje al jugador indicando que el territorio no está bajo el control del jugador.
	5. Se muestra un mensaje al jugador indicando que no dispone de suficientes recursos.

Tabla 57: Caso de uso CU-19

Identificador: CU-20	
Nombre	Ver recursos
Actor	Tiempo Cliente
Objetivo	Mostrar los recursos del jugador.
Precondiciones	El jugador está identificado en la aplicación.
Postcondiciones	La aplicación muestra el nivel del éter y hefestos del jugador.
Escenario básico	1. Se identifica el jugador en la aplicación. 2. Se muestra la información al jugador.
Escenario alternativo	

Tabla 58: Caso de uso CU-20

Identificador: CU-21	
Nombre	Ver población
Actor	Tiempo Cliente
Objetivo	Mostrar la población del jugador.
Precondiciones	El jugador está identificado en la aplicación.
Postcondiciones	La aplicación muestra el nivel población del jugador.
Escenario básico	1. Se identifica el jugador en la aplicación. 2. Se muestra la información al jugador.
Escenario alternativo	

Tabla 59: Caso de uso CU-21

Identificador: CU-22	
Nombre	Ver mejoras
Actor	Tiempo Cliente
Objetivo	Mostrar las mejoras del jugador.
Precondiciones	El jugador está identificado en la aplicación.
Postcondiciones	La aplicación muestra las mejoras del jugador.
Escenario básico	1. Se identifica el jugador en la aplicación. 2. Se muestra la información al jugador.
Escenario alternativo	

Tabla 60: Caso de uso CU-22

Identificador: CU-23	
Nombre	Ver información de jugador
Actor	Tiempo Cliente

Objetivo	Mostrar información sobre los recursos, mejoras y latencia del jugador.
Precondiciones	El jugador está identificado en la aplicación.
Postcondiciones	La aplicación muestra información sobre sus recursos y latencia al jugador.
Escenario básico	1. Se identifica el jugador en la aplicación. 2. Se muestra la información al jugador.
Escenario alternativo	

Tabla 61: Caso de uso CU-23

Identificador: CU-24	
Nombre	Ver información universo y sistemas
Actor	Tiempo Cliente
Objetivo	Mostrar actualizado el universo y los sistemas visibles para el jugador.
Precondiciones	El jugador está identificado en la aplicación.
Postcondiciones	La aplicación actualiza el universo y los sistemas visibles para el jugador.
Escenario básico	1. Se identifica el jugador en la aplicación. 2. Se actualiza la visión del universo y sistemas.
Escenario alternativo	

Tabla 62: Caso de uso CU-24

Identificador: CU-25	
Nombre	Ver información sistema y planetas
Actor	Tiempo Cliente
Objetivo	Mostrar actualizado el sistema y los planetas.
Precondiciones	Le está siendo mostrado un sistema al jugador.
Postcondiciones	La aplicación actualiza el sistema y los planetas que lo componen.
Escenario básico	1. Se accede a un sistema. 2. Se actualiza la visión del sistema y planetas.
Escenario alternativo	

Tabla 63: Caso de uso CU-25

Identificador: CU-26	
Nombre	Ver naves sistema
Actor	Tiempo Cliente
Objetivo	Mostrar actualizado las naves de batalla del sistema.
Precondiciones	Le está siendo mostrado un sistema al jugador.
Postcondiciones	La aplicación actualiza las naves de batalla del sistema.
Escenario básico	1. Se accede a un sistema. 2. Se actualizan las naves de batalla del sistema.
Escenario alternativo	

Tabla 64: Caso de uso CU-26

Identificador: CU-27	
Nombre	Ver secuencia sistema
Actor	Tiempo Cliente
Objetivo	Mostrar actualizado la secuencia de acciones de las naves de batalla del sistema.
Precondiciones	Le está siendo mostrado un sistema al jugador.
Postcondiciones	La aplicación actualizará la secuencia de acciones de las naves de batalla en el sistema.
Escenario básico	1. Se accede a un sistema. 2. Se actualiza la secuencia de acciones de las naves de batalla.

Escenario alternativo	

Tabla 65: Caso de uso CU-27

Identificador: CU-28	
Nombre	Ver información planeta
Actor	Tiempo Cliente
Objetivo	Mostrar actualizado el planeta.
Precondiciones	Le está siendo mostrado un planeta al jugador.
Postcondiciones	La aplicación actualiza el planeta.
Escenario básico	1. Se accede a un planeta. 2. Se actualiza la visión del planeta.
Escenario alternativo	

Tabla 66: Caso de uso CU-28

Identificador: CU-29	
Nombre	Ver naves planeta
Actor	Tiempo Cliente
Objetivo	Mostrar actualizado las naves de batalla del planeta.
Precondiciones	Le está siendo mostrado un planeta al jugador.
Postcondiciones	La aplicación actualiza las naves de batalla del planeta.
Escenario básico	1. Se accede a un planeta. 2. Se actualizan las naves de batalla del planeta.
Escenario alternativo	

Tabla 67: Caso de uso CU-29

Identificador: CU-30	
Nombre	Ver secuencia planeta
Actor	Tiempo Cliente
Objetivo	Mostrar actualizado la secuencia de acciones de las naves de batalla del planeta.
Precondiciones	Le está siendo mostrado un planeta al jugador.
Postcondiciones	La aplicación actualizará la secuencia de acciones de las naves de batalla en el planeta.
Escenario básico	1. Se accede a un planeta. 2. Se actualiza la secuencia de acciones de las naves de batalla.
Escenario alternativo	

Tabla 68: Caso de uso CU-30

Identificador: CU-31	
Nombre	Extraer recursos
Actor	Tiempo Servidor
Objetivo	Actualizar los recursos de los jugadores.
Precondiciones	
Postcondiciones	Los recursos de los jugadores se actualizan.
Escenario básico	1. Para cada jugador, sumar éter de aquellos territorios que estén bajo su control y tenga un extractor construido. De igual forma, sumar también hefesto de aquellos territorios que estén bajo su control y tengan una mina construida. 2. Se actualizan los recursos de cada jugador.
Escenario alternativo	

Tabla 69: Caso de uso CU-31

Identificador: CU-32	
Nombre	Recargar naves
Actor	Tiempo Servidor
Objetivo	Actualizar al máximo los movimientos y ataques de las naves de batalla de los jugadores.
Precondiciones	
Postcondiciones	Los movimientos y ataques de las naves de batalla son actualizados a su valor máximo.
Escenario básico	1. Para cada nave de batalla, actualizar el valor de sus movimientos y ataques al máximo en función del tipo de nave.
Escenario alternativo	

Tabla 70: Caso de uso CU-32

4.5. DIAGRAMA DE ESTADOS DE LA APLICACIÓN

Con los casos de uso, quedan descritas todas las interacciones posibles del usuario. No obstante, para que quede más claro el comportamiento de la aplicación se utilizará un diagrama de estados, que recoge de forma global los distintos estados posibles de la aplicación así como las transiciones entre ellos y los eventos que provocan estas transiciones:

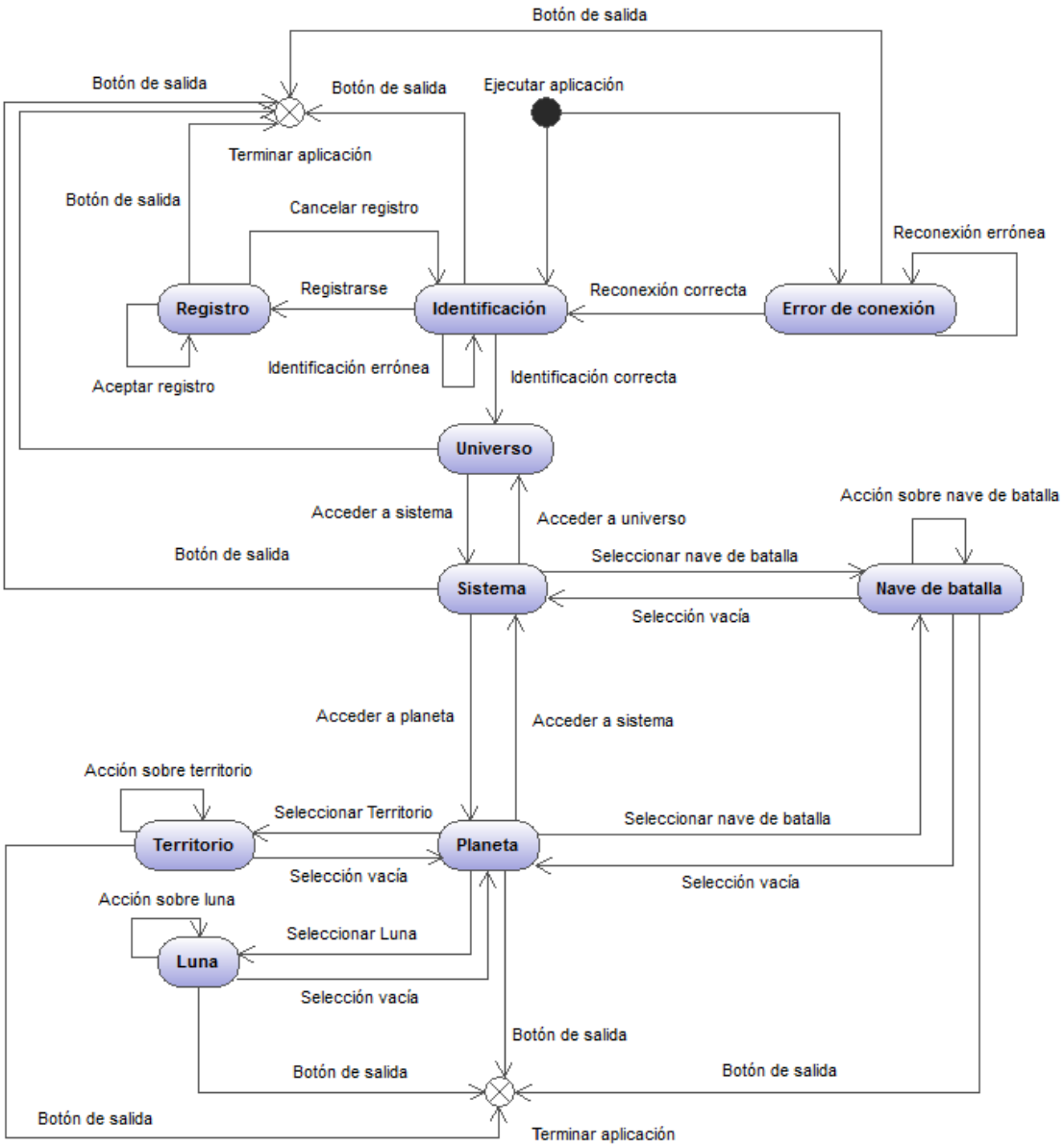


Tabla 71: Diagrama de estados

4.6. DEFINICIÓN DE REQUISITOS DE SOFTWARE

Los requisitos de software, orientados al programador y no al usuario, permiten definir de una forma más técnica y detallada las características y capacidades de la aplicación a desarrollar. Cuentan con los mismos campos a definir que los requisitos de usuario, y se describen con el mismo tipo de tabla, excepto que en este caso dos de los campos tendrán una consideración distinta:

- **Identificador:** código alfanumérico que identifica de forma unívoca cada requisito de software. La sintaxis del identificador se define de la siguiente forma:

  RS<tipo>-<número>

  - *RS:* hace referencia a que el requisito es de tipo “requisito de software”.

- *<Tipo>:* son dos caracteres que identifican el tipo de requisito de software, “FU” para requisitos funcionales, “IN” para requisitos de interfaz, “OP” para requisitos de operación, “RE” para requisitos de recursos y “SE” para requisitos de seguridad.
- *<Número>:* es una cifra de dos dígitos que identifica al requisito dentro de los requisitos de su mismo tipo. Empieza en 01 y crece una unidad por cada requisito.

Ejemplos: RSFU-05, RSSE-11, RSOP-03, etc.

- **Fuente:** requisito o requisitos de usuario en los que está basado este requisito de software.

Los requisitos software se clasifican en:

- **Funcionales:** definen “qué” debe hacer la aplicación.
- **Interfaz:** describen aspectos de la interfaz que permitirá la comunicación con el usuario.
- **Operación:** definen “cómo” va a realizar la aplicación sus funciones.
- **Recursos:** especifican los recursos básicos de los que depende la ejecución de la aplicación.
- **Comprobación:** indican las verificaciones que se llevan a cabo sobre los datos de entrada y de salida.
- **Documentación:** relativos a las ayudas proporcionadas al usuario dentro de la aplicación.
- **Seguridad:** indican los métodos de que dispone la aplicación para garantizar la seguridad los datos.
- **Portabilidad:** indican la adaptabilidad de la aplicación ante distintas plataformas de ejecución.

4.6.1. REQUISITOS FUNCIONALES

Identificador: RSFU-01			
Nombre	Identificación de jugador		
Fuente	RUC-01		
Descripción	Los jugadores pueden identificarse en la aplicación introduciendo su nombre de usuario y contraseña. Estos datos se envían al servidor para permitir el acceso en caso de que el jugador esté registrado y los datos recibidos sean correctos.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 72: Requisito funcional RSFU-01

Identificador: RSFU-02	
Nombre	Registro de jugadores
Fuente	RUC-02
Descripción	Los jugadores se registrarán en la aplicación suministrando los siguientes datos: • **Nombre de usuario:** Nombre de jugador usado para acceder a la aplicación de entre 3 y 32 caracteres. • **Email:** Dirección de correo electrónico del jugador. Mediante el uso de expresiones regulares se comprueba que esté bien formado. • **Contraseña:** Clave secreta del jugador usada para acceder a la aplicación. Es necesario introducirla dos veces durante el

	registro para confirmar que es correcta. • **Color Primario:** Primer color que identifica al jugador codificado en RGB de 24bits. • **Color Secundario:** Segundo color que identifica al jugador codificado en RGB de 24 bits. • **Bandera:** Imagen que utiliza el jugador para identificarse visualmente, se transforma a escala de grises y se ajusta a un tamaño de 40x40. En caso de no vincular ninguna imagen se presenta una bandera por defecto.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 73: Requisito funcional RSFU-02

Identificador: RSFU-03			
Nombre	Acceso a sistemas		
Fuente	RUC-03, RUC-07		
Descripción	El jugador puede acceder a los sistemas en los que se encuentren naves de batalla, territorios o lunas bajo su control. El acceso a un sistema puede ocasionar los siguientes mensajes de error: **"No Visible System":** Se produce si el jugador carece de territorios, naves de batalla o lunas bajos su control en el sistema indicado.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 74: Requisito funcional RSFU-03

Identificador: RSFU-04			
Nombre	Acceso a planetas		
Fuente	RUC-04, RUC-08		
Descripción	El jugador puede acceder a los planetas para interactuar con ellos.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 75: Requisito funcional RSFU-04

Identificador: RSFU-05			
Nombre	Construcción		
Fuente	RUC-03, RUC-09		

Descripción	Los jugadores podrán construir sobre territorios o lunas que estén bajo su control. Las edificaciones que pueden construir son las siguientes:
	
	• **Territorio:**
	  ○ **Ciudad:** Permite construir naves de batalla.
	  ○ **Laboratorio:** Permite realizar mejoras sobre las naves de batalla.
	  ○ **Extractor:** Permite obtener éter del territorio.
	  ○ **Mina:** Permite obtener hefestos del territorio.
	  ○ **Defensas:** Atacan de forma automática a los jugadores que intenten conquistar el territorio.
	• **Luna:**		

	○ **Base lunar:** Permite construir naves de batalla. La construcción de edificaciones puede ocasionar el siguiente error: • **"No Enough Resources":** Se produce si el jugador no dispone de los recursos necesarios.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 76: Requisito funcional RSFU-05

Identificador: RSFU-06			
Nombre	Crear naves de batalla		
Fuente	RUC-03, RUC-10, RUC-13		
Descripción	Los territorios con ciudades edificadas permiten crear naves de batalla que se ubicarán en uno de los espacios que sea adyacente al planeta y que no esté ocupado (o que esté ocupado por una nave del mismo tipo y jugador). Las lunas con bases lunares edificadas permiten crear naves de batalla del tipo pegassus que serán ubicadas en uno de sus espacios adyacentes y que no se encuentre ocupado (o que esté ocupado por una nave del mismo tipo y jugador). Los mensajes de error que puede ocasionar la creación de una nave de batalla son los siguientes: • **"No Enough Resources":** Se produce si el jugador no dispone de los recursos necesarios. • **"There is not free space for creating unit":** Se produce si la órbita del planeta está totalmente ocupada y no existen unidades del mismo jugador y tipo en ella. Los mensajes informativos indican la cantidad de recursos necesaria para cada tipo de nave de batalla.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 77: Requisito funcional RSFU-06

Identificador: RSFU-07			
Nombre	Mejora de naves de batalla		
Fuente	RUC-03, RUC-14		
Descripción	Los territorios que disponen de laboratorios edificados permiten mejorar el ataque y defensa de las naves de batalla. Una mejora supone un incremento en ataque o defensa respecto a su valor mínimo sin llegar a superar el máximo. El incremento seguirá una distribución logarítmica en la que el máximo nunca es alcanzado. Esta acción puede ocasionar los siguientes mensajes de error: **"No Enough Resources":** Se produce si el jugador no dispone de los suficientes recursos.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 78: Requisito funcional RSFU-07

Identificador: RSFU-08			
Nombre	Demonio de recarga de naves de batalla		
Fuente	RUC-11		
Descripción	El servidor dispondrá de un demonio [21] que realizará periódicamente la actualización de los ataques y movimientos de las naves de batalla.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	No Estable	**Verificabilidad**	Alta

Tabla 79: Requisito funcional RSFU-08

Identificador: RSFU-09			
Nombre	Demonio de recolección de recursos		
Fuente	RUC-15		
Descripción	El servidor dispondrá de un demonio que realizará periódicamente la actualización de los recursos de los jugadores.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	No Estable	**Verificabilidad**	Alta

Tabla 80: Requisito funcional RSFU-09

Identificador: RSFU-10			
Nombre	Movimiento de nave de batalla		
Fuente	RUC-03, RUC-12		
Descripción	Durante el movimiento el jugador indica el recuadro adyacente al que quiere mover la nave de batalla. Esta acción decrementa los movimientos de la nave de batalla del jugador. Esta acción puede ocasionar los siguientes mensajes de error: • **"No Enough Movements":** Se produce si la nave de batalla no dispone de los suficientes movimientos. • **"No possible Movement":** Se produce si el espacio indicado está ocupado.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 81: Requisito funcional RSFU-10

Identificador: RSFU-11	
Nombre	Ataque de nave de batalla
Fuente	RUC-03, RUC-12
Descripción	Durante el ataque el jugador indica la nave de batalla sobre la quiere dirigir sus ataques. Esta acción decrementa los ataques de la nave de batalla del jugador y disminuye la defensa y el número de la naves de batallas atacadas. La reducción de la defensa se calcula sumando proporcionalmente el ataque mínimo del atacante y la resta de las mejoras en ataque del atacante y las mejoras en defensa del atacado.

	Esta acción puede ocasionar los siguientes mensajes de error: • **"No Enough Attacks":** Se produce si la nave de batalla no dispone de los suficientes ataques. • **"No unit in this position":** Se produce si el espacio indicado está vacío.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 82: Requisito funcional RSFU-11

Identificador: RSFU-12			
Nombre	Salto de nave de batalla		
Fuente	RUC-03, RUC-12		
Descripción	Durante el salto el jugador indica el recuadro del sistema en el que quiere situar su nave de batalla. Esta acción reduce a cero los movimientos de la nave de batalla y la ubica en un sistema colindante al que ocupa dicha nave. Esta acción puede ocasionar los siguientes mensajes de error: • **"No Enough Movements":** Se produce si la nave de batalla no dispone de los suficientes movimientos. • **"No possible Jump":** Se produce si el espacio indicado se encuentra ocupado u obstaculizado.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 83: Requisito funcional RSFU-12

Identificador: RSFU-13			
Nombre	Conquistar		
Fuente	RUC-03, RUC-12		
Descripción	Durante la conquista el jugador indica el territorio o luna que se quiere conquistar. Esta acción reduce los ataques de la nave de batalla y hace controlar al jugador el territorio o luna en caso de ser satisfactorio. Las defensas del territorio atacan restando un valor constante multiplicado por su número a la defensa de las naves de batalla. Los territorios por su parte no pueden ser conquistados mientras dispongan de defensas. Esta acción puede ocasionar los siguientes mensajes de error: • **"No Enough Attacks":** Se produce si la nave de batalla no dispone de los suficientes ataques. • **"Planet is not formed":** Se produce si el territorio indicado pertenece a un planeta no geoformado.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 84: Requisito funcional RSFU-13

Identificador: RSFU-14			
Nombre	Geoformar		
Fuente	RUC-03, RUC-12		
Descripción	Durante la geoformación el jugador indica el planeta que se quiere geoformar. Esta acción la realizan las naves de batalla del tipo unicorn y reduce a cero los movimientos de las mismas. La acción puede realizarse únicamente sobre planetas no geoformados y hace que el jugador controle todos sus territorios. Esta acción puede ocasionar los siguientes mensajes de error: • **"No Enough Movements":** Se produce si la nave de batalla no dispone de los suficientes movimientos. • **"Planet formed yet":** Se produce si el planeta indicado se encuentra geoformado.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 85: Requisito funcional RSFU-14

Identificador: RSFU-15			
Nombre	Cambiar modo		
Fuente	RUC-12, RUR-19		
Descripción	El cambio de modo ocasiona que la nave de batalla pase del modo defensivo al ofensivo y viceversa. Durante el modo ofensivo la nave de batalla devolverá el ataque a aquellas naves de batalla que la ataquen. El modo defensivo o pasivo no realiza ninguna acción.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 86: Requisito funcional RSFU-15

Identificador: RSFU-16			
Nombre	Fin del juego		
Fuente	RUC-03, RUC-18		
Descripción	El juego informa al jugador sobre el fin del juego en caso de que no disponga de territorios en el universo. No se permite el acceso a los jugadores que han perdido el juego.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 87: Requisito funcional RSFU-16

4.6.2. REQUISITOS DE INTERFAZ

Identificador: RSIN-01	
Nombre	Aspecto visual de planetas
Fuente	RUC-16, RUR-04, RUR-14

Descripción	Los planetas no geoformados se representan como esferas con una textura característica. Por otra parte los planetas geoformados se representan como esferas que disponen de terreno, atmósfera y agua. El terreno cambia su textura en función de la proximidad a la estrella central, existen los siguientes tipo de terrenos: • **Terreno desértico:** Distancia a la estrella central corta. • **Terreno árido:** Distancia a la estrella central media-corta. • **Terreno forestal:** Distancia a la estrella central media -larga. • **Terreno ártico:** Distancia a la estrella central larga. El agua se distribuye por la superficie del planeta mediante un umbral y la salida de la función de ruido de Perlin sobre un espacio bidimensional, de forma que para una salida que oscila entre 0 y 255 los umbrales de agua van de 128 a 255. El valor mínimo crece proporcionalmente a la distancia entre el planeta y la estrella. El efecto de refracción y reflexión del agua se genera definiendo un Shader específico. La atmósfera está representada por una esfera translucida con una textura generada mediante la función de ruido de Perlin con salida bidimensional. Se utiliza únicamente el color blanco para aquellos pixeles que pasen de un terminado umbral.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	No Estable	**Verificabilidad**	Media

Tabla 88: Requisito de interfaz RSIN-01

Identificador: RSIN-02			
Nombre	Aspecto visual de estrellas		
Fuente	RUC-16, RUR-04, RUR-14		
Descripción	Las estrellas constituyen el foco de luz de todos los elementos situados en el sistema. Este efecto se consigue mediante la definición de un shader específico.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	No Estable	**Verificabilidad**	Media

Tabla 89: Requisito de interfaz RSIN-02

Identificador: RSIN-03			
Nombre	Aspecto visual de sistemas y universo		
Fuente	RUC-16, RUR-03, RUR-04, RUR-14		
Descripción	Los sistemas y el universo disponen de una nebulosa y de un campo de estrellas. Las nebulosas son generadas mediante ruido de Perlin con salida bidimensional y el campo de estrellas de forma aleatoria con una ocupación inferior al 10%.		
Prioridad	Media	**Necesidad**	Esencial
Estabilidad	No Estable	**Verificabilidad**	Media

Tabla 90: Requisito de interfaz RSIN-03

Identificador: RSIN-04	
Nombre	Mensajes de error
Fuente	RUC-03
Descripción	La interfaz de la aplicación muestra mensajes situados en el centro de

	la misma y que indican que el jugador ha realizado una acción no permitida.		
Prioridad	Media	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Media

Tabla 91: Requisito de interfaz RSIN-04

Identificador: RSIN-05			
Nombre	Mensajes de ayuda		
Fuente	RUC-04		
Descripción	La interfaz de la aplicación muestra mensajes discretos en la esquina inferior izquierda de la misma que facilitan al jugador la interacción con la aplicación.		
Prioridad	Media	**Necesidad**	Deseable
Estabilidad	Estable	**Verificabilidad**	Media

Tabla 92: Requisito de interfaz RSIN-05

Identificador: RSIN-06			
Nombre	Información de jugador		
Fuente	RUC-05		
Descripción	La interfaz muestra la latencia con el servidor, la cantidad de éter, hefestos y población del jugador cada 4 segundos. Cada uno de estos valores se calcula de la siguiente forma: • **Latencia:** Milisegundos transcurridos desde el envío de la petición hasta recibir la respuesta asociada. • **Éter:** Suma del éter proporcionado por cada uno de los territorio bajo el control del jugador y que disponen de un extractor construido. • **Hefestos:** Suma del hefestos proporcionado por cada uno de los territorio bajo el control del jugador y que disponen de una mina construida. • **Población:** Suma de la población proporcionada por cada territorio bajo el control del jugador.		
Prioridad	Media	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Media

Tabla 93: Requisito de interfaz RSIN-06

Identificador: RSIN-07			
Nombre	Información de los elementos del escenario		
Fuente	RUC-06		
Descripción	La interfaz muestra en la parte inferior derecha información acerca de los elementos del escenario con los que el jugador interactúa.		
Prioridad	Media	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Media

Tabla 94: Requisito de interfaz RSIN-07

Identificador: RSIN-08	
Nombre	Mostrar recursos necesarios
Fuente	RUC-04, RUR-12
Descripción	La interfaz muestra al jugador los recursos necesarios para construir

	una edificación o nave de batalla determinada.		
Prioridad	Media	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Media

Tabla 95: Requisito de interfaz RSIN-08

Identificador: RSIN-09			
Nombre	Aspecto visual de naves de batalla		
Fuente	RUC-10, RUR-11		
Descripción	Las naves de batalla deberán disponer de un modelo propio que las represente. Además deberán tener detalles con los colores primario y secundario del jugador al que pertenecen, ya que de esta forma son más fácilmente reconocibles.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 96: Requisito de interfaz RSIN-09

Identificador: RSIN-10			
Nombre	Efectos de naves de batalla		
Fuente	RUC-12, RUR-11		
Descripción	Las naves de batalla utilizan un sistema de partículas para representar la estela que ocasiona su movimiento y para representar los disparos que realizan.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 97: Requisito de interfaz RSIN-10

Identificador: RSIN-11			
Nombre	Vista de planetas		
Fuente	RUC-08		
Descripción	La interacción con los planetas debe facilitarse con una vista que permita girar en torno a los mismos, siendo de esta forma más fácilmente accesibles todos los territorios del mismo.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 98: Requisito de interfaz RSIN-11

Identificador: RSIN-12			
Nombre	Visualización de acciones de jugadores		
Fuente	RUC-17		
Descripción	Los jugadores observan las acciones de movimiento, conquista, ataque y salto realizadas por las naves de batalla del resto de jugadores.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 99: Requisito de interfaz RSIN-12

Identificador: RSIN-13			
Nombre	Acciones de los elementos del escenario		

Fuente	RUC-07, RUC-08, RUC-09, RUC-10, RUC-12, RUC-13, RUC-14		
Descripción	Las acciones sobre los elementos del escenario se realizan mediante botones representados mediante la librería Windows.Forms.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 100: Requisito de interfaz RSIN-13

Identificador: RSIN-14			
Nombre	Cámara de los escenarios		
Fuente	RUR-08, RUR-09, RUR-10		
Descripción	La cámara varía su comportamiento en función del escenario representado. Sus características en función del escenario son las siguientes: • **Universo:** Forma un ángulo de 90º con el plano sobre el que se ubican los elementos del escenario. Su movimiento se realiza a una altura constante sobre el mismo plano y haciendo uso de las teclas direccionales del teclado. • **Sistemas:** Forma un ángulo de 45º con el plano sobre el que se ubican los elementos del escenario. Su movimiento se realiza a una altura constante sobre el mismo plano y haciendo uso de las teclas direccionales. La cámara reduce y amplía su distancia al plano mediante el uso de la rueda del ratón y gira la dirección del objetivo en ángulos de 45º mediante la pulsación de las teclas Avpág y Repág. • **Planeta:** Orbita sobre el planeta a una distancia constante de su centro. El movimiento de la cámara se realiza pulsando las teclas de dirección. La cámara reduce su distancia al centro del planeta mediante el uso de la rueda del ratón.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 101: Requisito de interfaz RSIN-14

4.6.3. REQUISITOS DE OPERACIÓN

Identificador: RSOP-01	
Nombre	Creación de sistemas
Fuente	RUC-02, RUC-16, RUR-03, RUR-14
Descripción	La aplicación genera los sistemas del universo en función de su demanda (a partir del registro de jugadores). El universo se expande mediante la creación de sistemas que se crean alrededor de los ya generados. La expansión se produce en anillos de cada vez mayor radio cuyo origen es un sistema inicial ubicado en el centro del universo.

Prioridad	Media	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Media

Tabla 102: Requisito de operación RSOP-01

Identificador: RSOP-02			
Nombre	Distribución de planetas		
Fuente	RUC-16, RUR-04, RUR-14		
Descripción	Los sistemas que disponen de planetas los ubican alrededor de la estrella central. Su situación es un punto aleatorio de la circunferencia cuyo centro es la posición de la estrella y cuyo diámetro itera aumentando su valor hasta alcanzar la longitud del sistema.		
Prioridad	Media	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Media

Tabla 103: Requisito de operación RSOP-02

Identificador: RSOP-03			
Nombre	Características de planetas		
Fuente	RUC-16, RUR-05, RUR-14		
Descripción	Los planetas son de los siguientes tamaños: • **Grande:** Con diámetro 4 (cuadrados de espacio mínimo) y 64 territorios. • **Pequeño:** Con diámetro 2 (cuadrados de espacio mínimo) y 8 territorios.		
Prioridad	Media	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Media

Tabla 104: Requisito de operación RSOP-03

Identificador: RSOP-04			
Nombre	Características de territorios		
Fuente	RUC-16, RUR-06, RUR-14		

Descripción	Los territorios están delimitados por cubos de longitud constante distribuidos a lo largo del volumen de la esfera. Cada uno de ellos dispone de las siguientes características:
	• **Éter:** Nivel de éter que proporciona el territorio en caso de que sea construido un extractor en él. El valor es constante y calculado aleatoriamente entre 127 y 255.
	• **Hefestos:** Nivel de hefestos que proporciona el territorio en caso de que sea construida una mina en él. El valor es constante y calculado aleatoriamente entre 127 y 255.
	• **Población:** Nivel de población que proporciona el territorio, el cálculo del valor varía en función del tipo de planeta:
	  ○ **Grande:** Entre 1 y 2.
	  ○ **Pequeño:** Entre 3 y 4.		

Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Media

Tabla 105: Requisito de operación RSOP-04

Identificador: RSOP-05	
Nombre	Distribución agujeros negros y campos de meteoritos

Fuente	RUC-16, RUR-03, RUR-04, RUR-14		
Descripción	Los agujeros negros del universo y los campos de meteoritos de los sistemas se distribuyen aleatoriamente y ocupando completamente los recuadros de espacio mínimo. Su ocupación no debe de superar el 10 % del espacio libre de los sistemas y del universo.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Media

Tabla 106: Requisito de operación RSOP-05

Identificador: RSOP-06			
Nombre	Características de construcciones		
Fuente	RUR-12		

Descripción	A continuación se exponen las características de las construcciones que es posible construir en los territorios y lunas que están bajo el control del jugador:
	• **Mina:**
	  ○ **Éter:** 2000.
	• **Extractor:**
	  ○ **Hefestos**: 4000.
	• **Laboratorio:**
	  ○ **Éter:** 20000.
	  ○ **Hefestos:** 10000.
	• **Ciudad:**
	  ○ **Éter:** 6000.
	  ○ **Hefestos:** 12000.
	• **Defensa:**
	  ○ **Éter:** 400.
	  ○ **Hefestos:** 400.
	• **Base lunar:**
	  ○ **Éter:** 4000.
	  ○ **Hefestos:** 8000.		

Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 107: Requisito de operación RSOP-06

Identificador: RSOP-07			
Nombre	Tipos y características de naves de batalla		
Fuente	RUC-10		

Descripción	A continuación se exponen las características de las naves de batalla que es posible crear en los territorios que disponen de ciudades edificadas:
	• **Pegasus:**
	  ○ **Ataque mínimo:** 10.
	  ○ **Ataque máximo:** 15:
	  ○ **Defensa:** 50.
	  ○ **Número de movimientos:** 3.
	  ○ **Número de ataques:** 3.
	  ○ **Población:** 2.
	  ○ **Éter:** 200.
	  ○ **Hefestos:** 400.
	• **Unicorn:**
	  ○ **Ataque mínimo:** 5.		

	○ **Ataque máximo:** 10: ○ **Defensa:** 100. ○ **Número de movimientos:** 3. ○ **Número de ataques:** 2. ○ **Población:** 4. ○ **Éter:** 400. ○ **Hefestos:** 400. • **Chimera:** ○ **Ataque mínimo:** 15. ○ **Ataque máximo:** 30: ○ **Defensa:** 300. ○ **Número de movimientos:** 5. ○ **Número de ataques:** 8. ○ **Población:** 8. ○ **Éter:** 600. ○ **Hefestos:** 800.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 108: Requisito de operación RSOP-07

Identificador: RSOP-08			
Nombre	Utilización de recursos		
Fuente	RUR-01, RUR-12		
Descripción	Los jugadores pueden hacer uso de tres tipos de recursos en los siguientes acciones: • **Hefestos:** Creación de naves de batalla, mejora de naves de batalla, construcción de edificios. • **Éter:** Creación de naves de batalla, mejora de naves de batalla, construcción de edificios. • **Población:** Creación de naves de batalla.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 109: Requisito de operación RSOP-08

Identificador: RSOP-09			
Nombre	Espacio mínimo del escenario		
Fuente	RUR-02		
Descripción	Las dimensiones del universo, de los sistemas, de los planetas, de los territorios y de las lunas representadas en la aplicación son divisibles por un espacio mínimo con forma cuadrada. Este espacio mínimo conforma las áreas de interacción de la aplicación.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 110: Requisito de operación RSOP-09

Identificador: RSOP-10	
Nombre	Elementos del escenario
Fuente	RUR-03, RUR-04, RUR-05, RUR-06
Descripción	Existen tres tipos de escenarios que se componen de los siguientes elementos:

	• **Universo:** Compuesto por agujeros negros y sistemas. • **Sistemas:** Compuestos por campos de meteoritos y algunos de ellos por una estrella, planetas y lunas. Las naves de batalla también se representan en este escenario. • **Planetas:** Compuestos por territorios y lunas. Las naves de batalla también se representan en este escenario.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 111: Requisito de operación RSOP-10

Identificador: RSOP-11			
Nombre	Interacción con elementos del escenario		
Fuente	RUR-07		
Descripción	La interacción con los elementos del escenario se realiza con el ratón y la pulsación del botón izquierdo del mismo. La identificación del elemento seleccionado se calcula comprobando que las coordenadas del ratón durante la pulsación, se encuentren dentro de la circunferencia formada por la proyección en la pantalla de la esfera mínima que contiene al elemento.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 112: Requisito de operación RSOP-11

Identificador: RSOP-12			
Nombre	Visibilidad del jugador		
Fuente	RUR-13		
Descripción	No se representan aquellos elementos de los escenarios que no son visibles para el jugador. Únicamente son visibles aquellos sistemas que dispongan de naves de batalla, territorios o lunas bajo el control del jugador y los adyacentes a los descritos.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 113: Requisito funcional RSOP-12

Identificador: RSOP-13			
Nombre	Base de datos		
Fuente	RUR-16		
Descripción	El estado del juego se almacena haciendo uso de una base de datos relacional MySQL.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 114: Requisito de operación RSOP-13

Identificador: RSOP-14			
Nombre	Concurrencia de datos		
Fuente	RUR-16, RUR-18		
Descripción	La consistencia de los datos se asegura mediante el uso de sentencias de bloqueo de tuplas ofrecidas por MySQL.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 115: Requisito de operación RSOP-14

Identificador: RSOP-15			
Nombre	Concurrencia de procesos		
Fuente	RUR-16, RUR-18		
Descripción	Los problemas de sincronización entre los procesos del servidor se resuelve haciendo uso de los elementos de sincronización ofrecidos por la librería System.Threading.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 116: Requisito de operación RSOP-15

4.6.4. REQUISITOS DE RECURSOS

Identificador: RSRE-01			
Nombre	Culling de los elementos del escenario		
Fuente	RUR-17		
Descripción	Se aplica la técnica de optimización en el renderizado Frustum Culling [22] en todos los escenarios de la aplicación.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 117: Requisito de recursos RSRE-01

4.6.5. REQUISITOS DE SEGURIDAD

Identificador: RSSE-01			
Nombre	Identificación segura		
Fuente	RUC-01, RUR-15		
Descripción	En la identificación de jugador se cifra el resultado de aplicar la función SHA-512 sobre la contraseña.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 118: Requisito de seguridad RSSE-01

Identificador: RSSE-02			
Nombre	Negociación de clave de sesión		
Fuente	RUR-15		
Descripción	A iniciar la aplicación se negocia la clave de sesión entre cliente y servidor utilizando el protocolo Diffie-Hellman. El servidor almacenará la clave de sesión utilizando cookies.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 119: Requisito de seguridad RSSE-02

Identificador: RSSE-03

Nombre	Seguridad de las peticiones		
Fuente	RUR-15		
Descripción	Los envíos del cliente al servidor se realizan mediante el verbo POST de HTTP donde los campos están cifrados con el algoritmo Rijndael. Para garantizar la integridad y autenticación, se incluye el campo HASH cuyo valor es el resultado de la función SHA-512 sobre el mensaje concatenado con la contraseña de usuario.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 120: Requisito de seguridad RSSE-03

Identificador: RSSE-04			
Nombre	Seguridad de las respuestas		
Fuente	RUR-15		
Descripción	Los envíos del servidor al cliente se realizan cifrando la respuesta XML mediante el algoritmo Rijndael. Se incluye además el resultado de la función SHA-512 sobre el mensaje concatenado con la contraseña para garantizar la integridad y autenticación.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 121: Requisito de seguridad RSSE-04

Identificador: RSSE-05			
Nombre	Almacenamiento seguro de contraseñas		
Fuente	RUR-15, RUR-16		
Descripción	No se almacena información sensible en claro como lo son las contraseñas de los jugadores, sino que se almacena el resultado de la función SHA-512 sobre estas.		
Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 122: Requisito de seguridad RSSE-05

Identificador: RSSE-06			
Nombre	Acceso seguro a la base de datos		
Fuente	RUR-15, RUR-16		
Descripción	La base de datos dispone de un único usuario con contraseña habilitada y que dispone solo de los permisos precisos para desempeñar su labor.		
Prioridad	Alta	**Necesidad**	Deseable
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 123: Requisito de seguridad RSSE-06

Identificador: RSSE-07	
Nombre	Captura de excepciones
Fuente	RUR-18
Descripción	El servidor captura todas las excepciones que puedan surgir durante la ejecución de la aplicación y además almacenará información relativa a cada una de ellas para su posterior análisis y tratamiento.

Prioridad	Alta	**Necesidad**	Esencial
Estabilidad	Estable	**Verificabilidad**	Alta

Tabla 124: Requisito de seguridad RSSE-07

5. DISEÑO

A partir del análisis realizado en el anterior capítulo, se lleva a cabo la fase de diseño de la aplicación que describirá en detalle cómo va a ser implementada.

Con este fin, este capítulo recoge los siguientes diagramas comentados: Diagrama de subsistemas de la aplicación, diagrama de despliegue, diagrama de las principales clases de la aplicación, una descripción de las clases mostradas y diagramas del diseño de la base de datos (modelo entidad-relación y modelo lógico).

Para todos los tipos de diagramas mencionados, a excepción del de despliegue, se realizará la distinción entre los que pertenecen al cliente y los que pertenecen al servidor.

5.1. DIAGRAMAS DE SUBSISTEMAS Y COMPONENTES

Los diagramas de cliente y servidor presentados a continuación se inspiran en la arquitectura clásica **MVC** [23] en la que cliente y servidor se comunican por medio de sus respectivos subsistemas de comunicación, gestionan su lógica en sus respectivos controladores y donde el cliente se encarga del subsistema vista (representación gráfica e interacción) y el servidor del modelo (persistencia de datos).

5.1.1. CLIENTE

La aplicación cliente se divide en tres grandes subsistemas que definen a alto nivel los principales módulos de los que está compuesta, subdivididos al mismo tiempo en varios niveles de componentes:

- **View**: Recoge la definición de todos los elementos con representación gráfica tridimensional mediante el componente **DrawableElements** y con **Windows.Forms** todas las interfaces, formularios y elementos de interacción con el jugador.
- **ClientCommunication:** Define la comunicación de la aplicación entre cliente y servidor, el desaplanado de los paquetes recibidos (**Unmarshalling**) y la seguridad de las comunicaciones (**Security**).
- **ClientController:** Gestiona la lógica, animación y comportamiento de los elementos gráficos y la interacción del jugador con los mismos mediante el componente **InteractiveElements**.

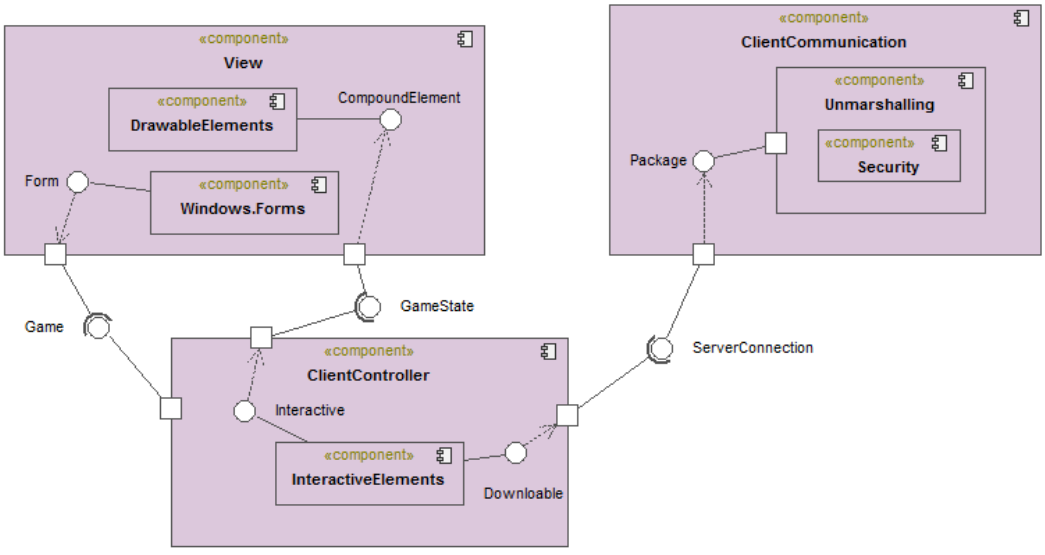


Ilustración 24: Diagrama de subsistemas y componentes del cliente

5.1.2. SERVIDOR

La aplicación servidor queda definida mediante tres grandes subsistemas:

- **ServerController:** Define la lógica y reglas de juego y gestiona la concurrencia que implica, además mediante el componente **ScenarioGeneration** define la generación dinámica de escenarios y los diversos elementos contenidos en ellos.
- **Model:** Recoge la persistencia y consulta de la información en la base de datos y gestiona la concurrencia a nivel de datos mediante el uso de transacciones.
- **ServerCommnucation:** Define la comunicación de la aplicación entre servidor y cliente, el aplanado de los paquetes enviados (M**arshalling**) y la seguridad de las comunicaciones (**Security**).

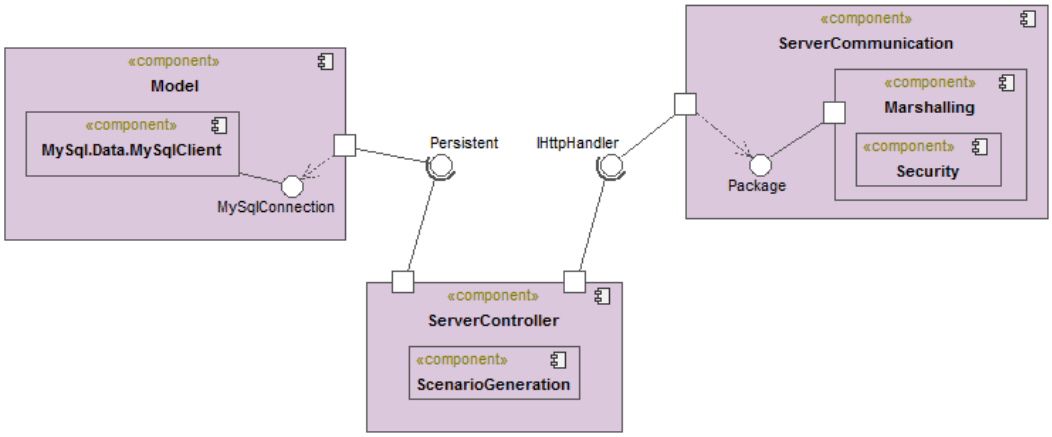


Ilustración 25: Diagrama de subsistemas y componentes del servidor

5.2. DIAGRAMA DE DESPLIEGUE

El diagrama de despliegue permite representar la disposición de los elementos hardware que son necesarios para ejecutar la aplicación, así como los diversos entornos de ejecución y artefactos software requeridos para este fin.

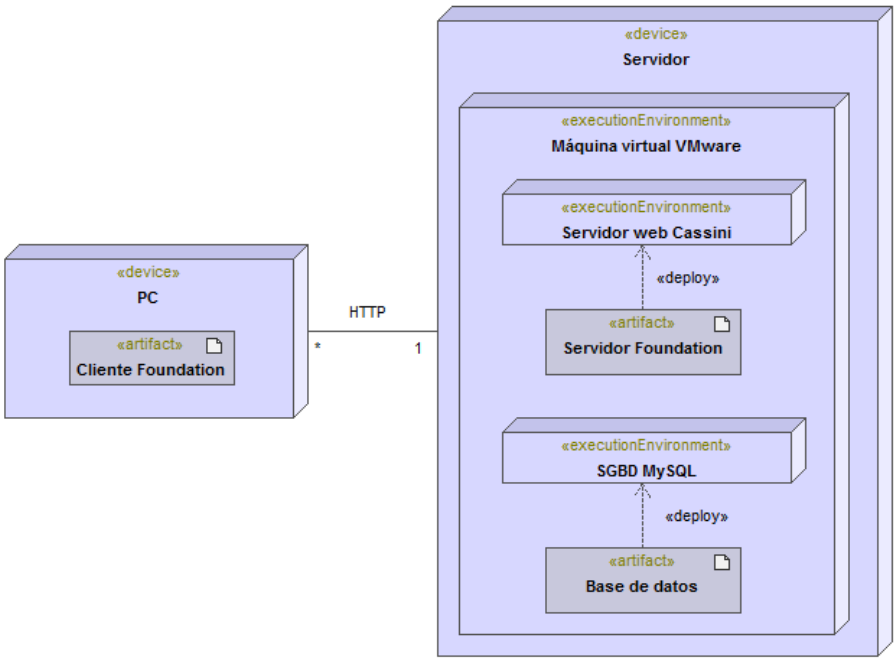


Ilustración 26: Diagrama de despliegue

En primera instancia se dispone de un ordenador personal por cada uno de los clientes (con Sistema Operativo Windows XP o superior instalado) sobre el que se despliega la aplicación cliente. Mediante tráfico HTTP se comunican cliente y servidor formando una relación de uno a varios. El Servidor por su parte dispone del software de virtualización VMware con el que se despliega y se hace uso de una máquina virtual con sistema operativo Windows Server 2003. En dicha máquina se encuentra ejecutándose el servidor web Cassini y el sistema gestor de bases de datos MySQL, la aplicación servidor y la base de datos se despliegan respectivamente en los entornos mencionados.

5.3. DIAGRAMA DE CLASES

El diagrama de clases describe la estructura de una aplicación mostrando las clases con métodos y atributos, y las relaciones entre ellas. Para este caso particular no se mostrarán exhaustivamente todas las clases de las que dispone la aplicación, ya que las clases inferiores de las jerarquías de funcionalidad similar no serán incluidas por falta de espacio. Por el mismo motivo no se mostrarán los atributos y métodos si no que se explicarán aquellos que se consideren más relevantes para comprender la aplicación.

5.3.1. CLIENTE

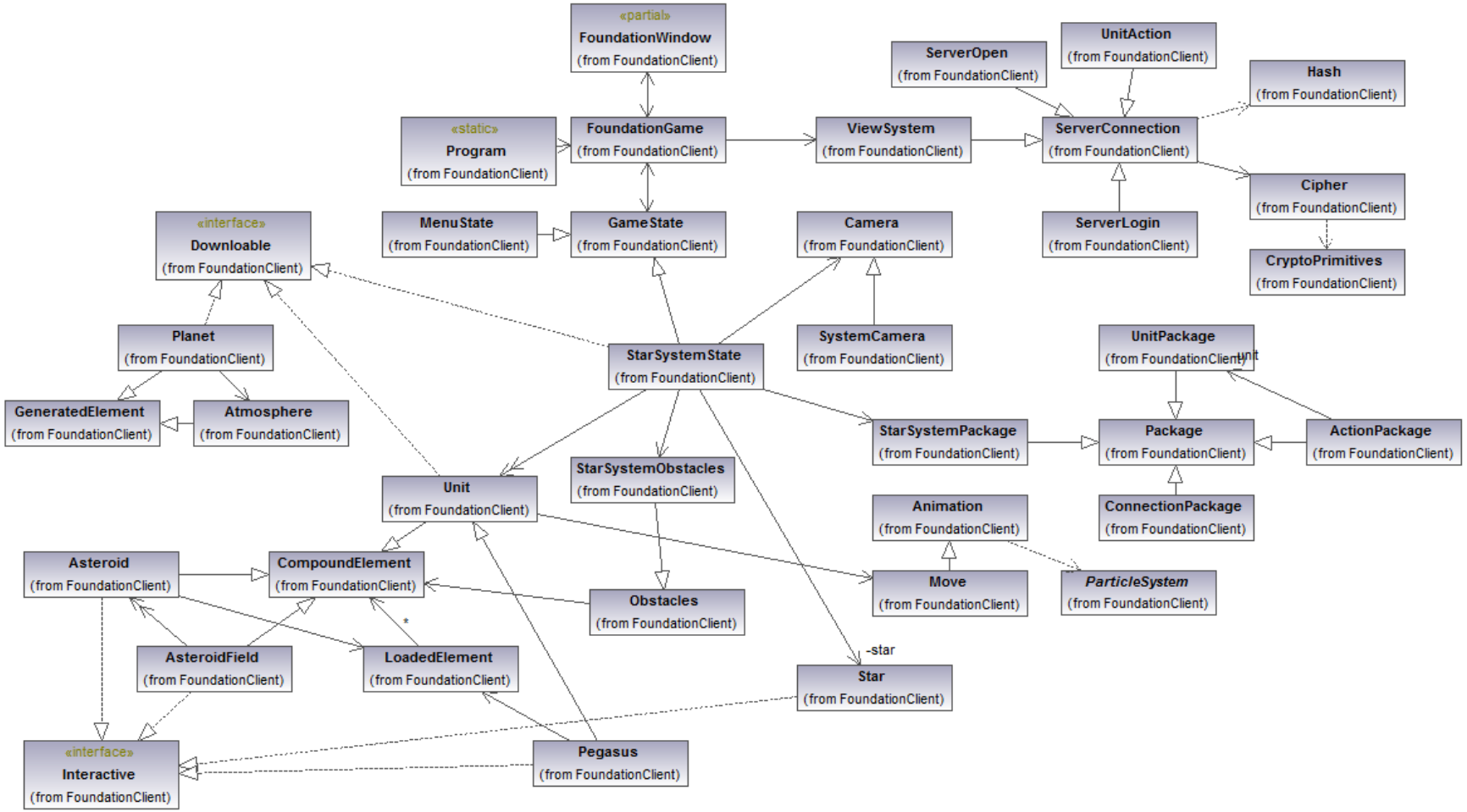


Ilustración 27: Diagrama de clases del cliente

5.3.1.1. JERARQUÍA SERVERCONNECTION

La jerarquía está diseñada con el fin de lograr comunicar el cliente con el servidor. Todas las peticiones que se realizan al servidor y sus consecuentes respuestas heredan de la clase **ServerConnection**.

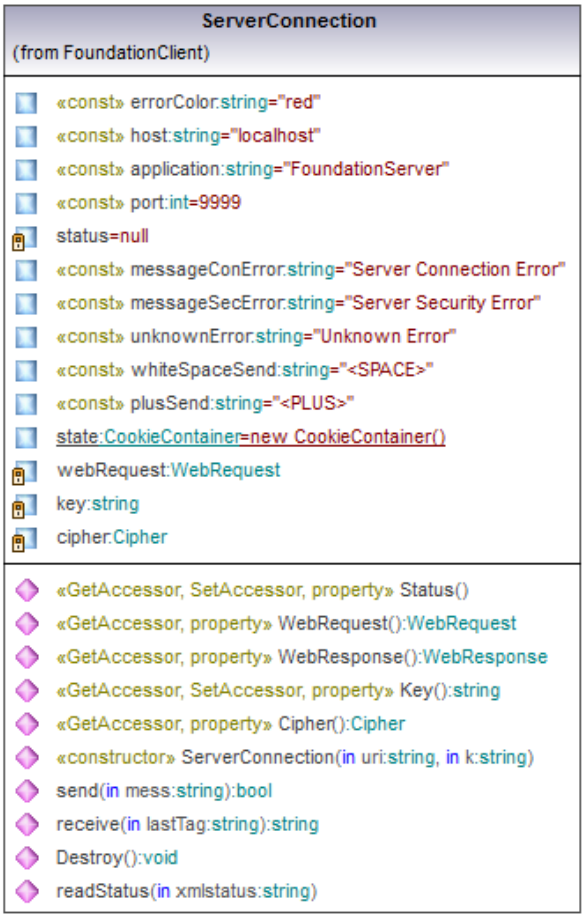


Ilustración 28: Clase ServerConnection

La clase contiene toda la información necesaria para comunicar al cliente con el servidor mediante el protocolo HTTP y haciendo uso de la clase **WebRequest**. Además define los métodos de envío y recepción (**send** y **receive**) que deberán utilizar las clases hijas para comunicarse y transformar debidamente los datos recibidos. En caso de que la comunicación a nivel HTTP falle, podrá recuperarse el error haciendo uso del método **readStatus**. El cifrado y descifrado de los mensajes se realiza por medio de la clase **Cipher** que implementa un cifrador de tipo Rijndael.

5.3.1.2. JERARQUÍA PACKAGE

El desaplanado de las respuestas recibidas del servidor se implementa mediante clases que heredan de **Package**.

Ilustración 29: Clase Package del cliente

Mediante la clase **XmlReader** se tratan los mensajes recibidos del servidor en formato XML. La clase **Package** implementa las funciones de lectura de tipos básicos, las clases herederas deberán usarlos para interpretar tipos de datos más complejos (compuestos a partir de estos).

5.3.1.2. PATRÓN DE DISEÑO STATE APLICADO EN LA JERARQUÍA GAMESTATE

La jerarquía **GameState** implementa el patrón de diseño State [24] y es utilizado para transitar entre los diversos estados de la aplicación cliente. Los estados del juego se definen mediante clases que tienen compartamiento similar y que heredan de **GameState**. El juego consta de los siguientes estados/clases:

- **MenuState:** Muestra la interfaz principal al jugador y carece de representación tridimensional.
- **UniverseState:** Muestra el universo al jugador y los elementos que lo conforman (sistemas y agujeros negros).
- **SystemState:** Muestra un sistema al jugador y los elementos que lo conforman (Planetas, lunas, estrellas, campos de meteoritos y naves de batalla).
- **PlanetState:** Muestra un planeta al jugador y las lunas y naves de batalla que se encuentran en su órbita.

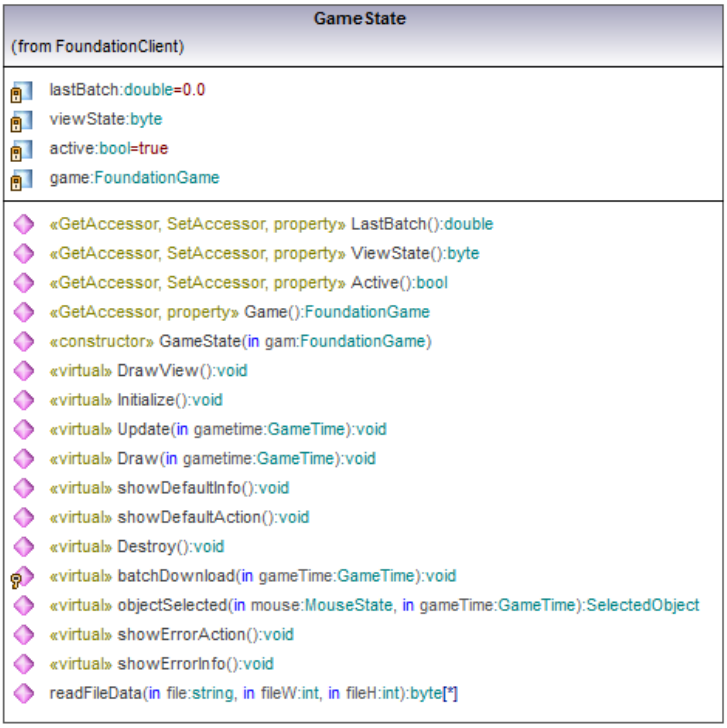


Ilustración 30: Clase GameState del cliente

Se definen los métodos necesarios para actualizar el estado (**Update**) y para actualizar su contenido respecto al servidor de forma periódica (**batchDownload**). El resto de métodos son utilizados para mostrar información por defecto en caso de que no exista ningún elemento seleccionado o para mostrar mensajes de error.

5.3.1.3. REPRESENTACIÓN MEDIANTE GENERATEDELEMENT Y COMPOUNDELEMENT

La representación tridimensional de los elementos que conforman los diversos escenarios de juego, hace uso de las clases **GeneratedElement** y **CompoundElement**. La primera de ellas se reserva para aquellos elementos generados dinámicamente por el servidor y que deben ser construidos a partir de la información recibida. Los algoritmos de generación son diversos y dependen del elemento que quiera ser generado (Atmósfera, planeta, etc.).

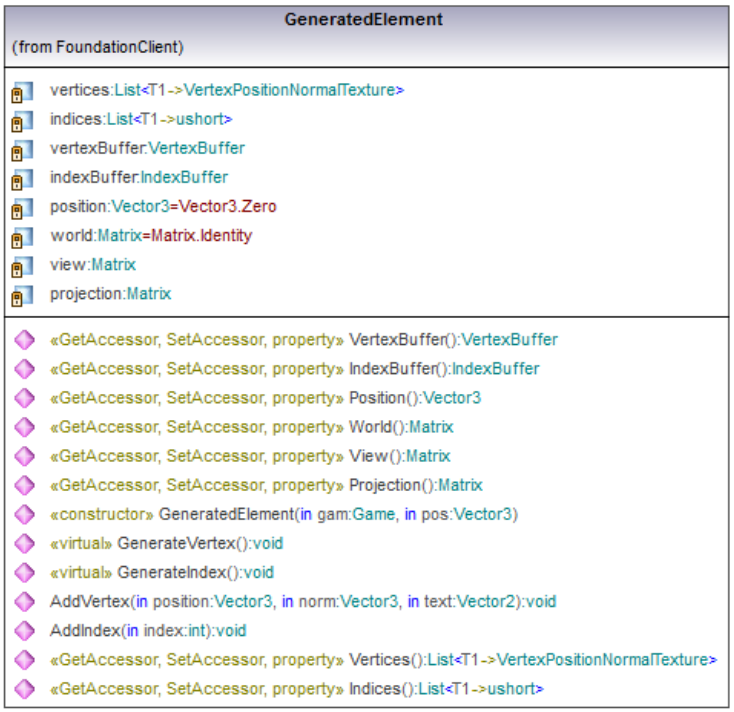


Ilustración 31: Clase GeneratedElement del cliente

La clase contiene todos los elementos necesarios para su representación tridimensional (Matrices de proyección, vista y mundo). Además ofrece métodos para generar los vértices e índices de la maya tridimensional que representará al elemento en pantalla (**GenerateVertex** y **GenerateIndex**) y que las clases que hereden de ella deberán implementar de acuerdo a sus necesidades.

Por su parte, **CompoundElement** está destinada a los elementos cuya representación gráfica se encuentre localmente almacenada en forma de modelo tridimensional (Formato DirectX [25]). El elemento en cuestión puede estar formado por una colección de modelos que serán identificados en la aplicación por la clase **LoadedElement**.

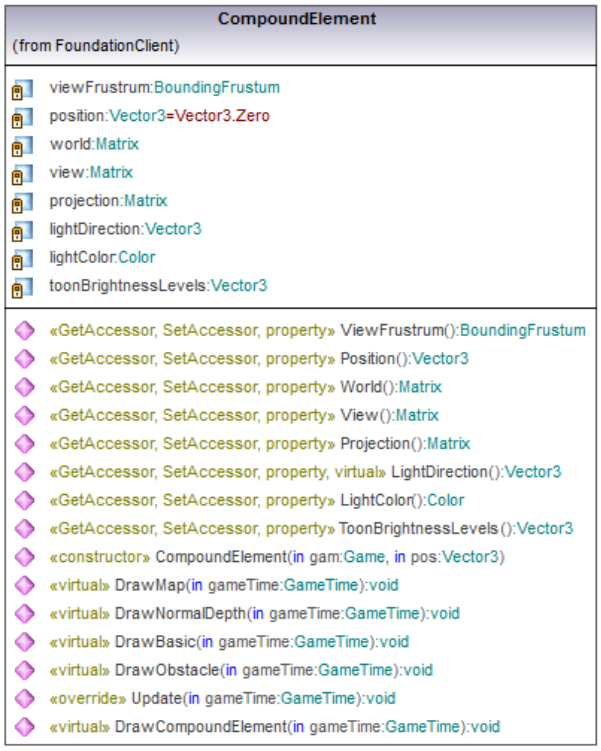


Ilustración 32: Clase CompoundElement del cliente

De forma análoga a la anterior, la clase dispone de los atributos requeridos para su representación gráfica y actualización. La principal diferencia es que en esta se definen métodos para poder utilizar diversos efectos con los modelos escogidos (**DrawMap**, **DrawNormalDepth**, **DrawObstacle** y **DrawBasic**).

5.3.1.4. INTERFACES DOWNLOABLE E INTERACTIVE

El cliente dispone de dos interfaces que las clases podrán implementar en caso de que requieran cumplir alguna de las funcionalidades ofrecidas. La interfaz **Downloable** se utilizará para aquellas clases que requieran ser actualizadas con información almacenada en el servidor y la interfaz **Interactive** para aquellos elementos cuya representación gráfica disponga de interacción con el jugador (selección con el ratón o mediante la interfaz).

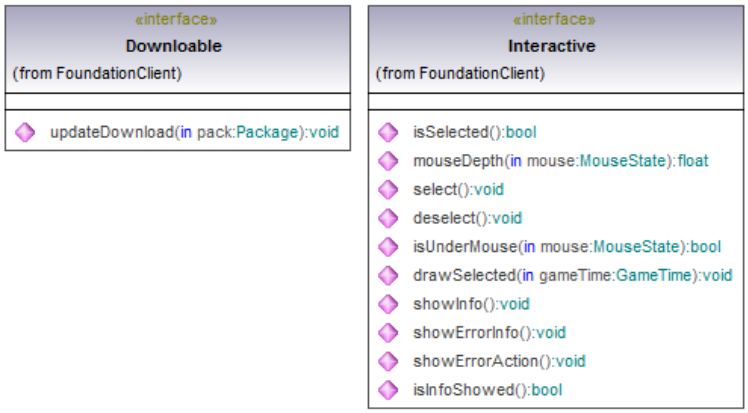


Ilustración 33: Interfaces Downloable e Interactive

Mientras **Downloable** únicamente ofrece un método en el que se implementará la comunicación y actualización del elemento, **Interactive** ofrece métodos de interacción con el ratón (**IsUnderMouse**), de representación en caso de que el elemento se encuentre seleccionado (**DrawSelected**), destinados a mostrar información sobre el elemento (**ShowErrorInfo**, **showErrorAction**) y de lógica de interacción.

5.3.2. SERVIDOR

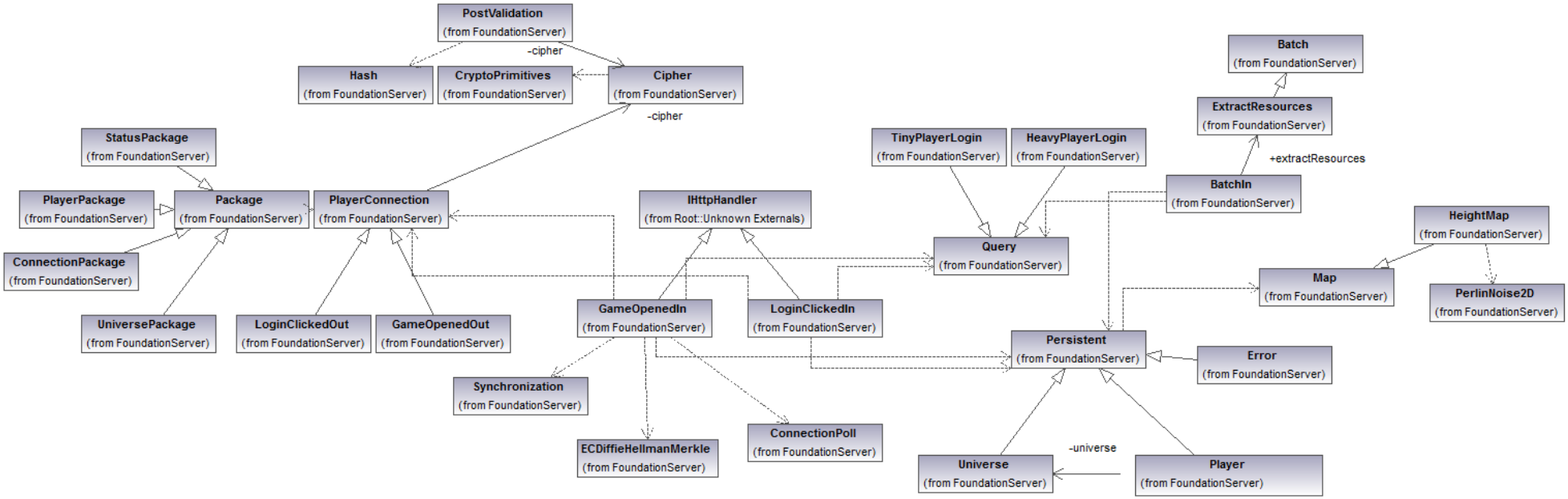


Ilustración 34: Diagrama de clases del servidor

5.3.2.1. JERARQUÍA PLAYERCONNECTION

La finalidad del diseño de esta jerarquía es la de proporcionar un mecanismo de comunicación entre servidor y cliente. Todas las respuestas que el servidor envíe al cliente serán definidas como clases que hereden de **PlayerConnection**.

Ilustración 35: Clase PlayerConnection del servidor

La clase define un método de envío (**send**) que recibe de entrada un texto en formato XML y que hace uso de la clase **HTTPResponse** para realizar el envío mediante el protocolo HTTP. La codificación de los mensajes se realiza utilizando el objeto **Cipher** (idéntico al de la aplicación cliente).

5.3.2.2. JERARQUÍA PACKAGE

Las clases que hereden de la clase **Package** están destinadas a aplanar las entidades tratadas por la aplicación y que posteriormente serán enviadas al cliente para responder a una petición determinada.

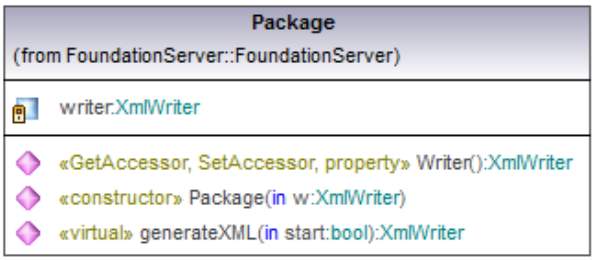


Ilustración 36: Clase Package del servidor

La clase ofrece únicamente un método que hace uso de la clase **XmlWriter** para transformar la entidad especificada en una cadena de texto con formato XML. Las clases que hereden de **PlayerConnection** harán uso de este tipo de objetos para transformar los datos y que puedan ser debidamente codificados.

5.3.2.3. JERARQUÍA PERSISTENT Y QUERY

La manipulación de la base de datos se hace por medio de dos tipos de objetos, aquellos que heredan de la clase **Persistent** y **Query**. Los objetos que heredan de la clase **Persistent** adquieren la semántica de persistentes en la base de datos.

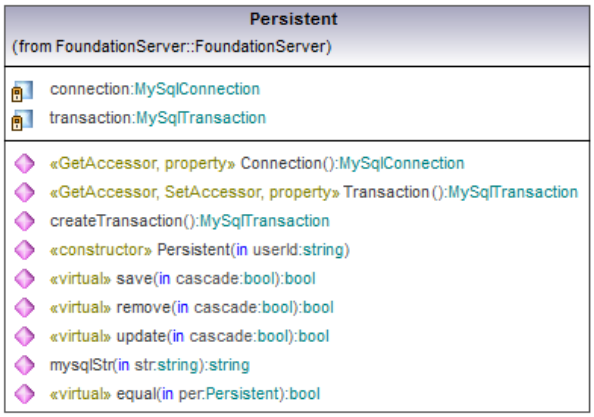


Ilustración 37: Clase Persistent del servidor

Además de métodos de métodos de manipulación de datos (**save**, **remove** y **update**), se ofrecen métodos para comparar elementos persistentes (**equal**) y para crear transacciones seguras (**createTransaction**) que las clases herederas deberán implementar.

Por otra parte, los objetos que heredan de la clase **Query** están destinados a realizar consultas sobre la base de datos de un determinado tipo de objeto. Serán por tanto los encargados de obtener información que fue almacenada de forma persistente.

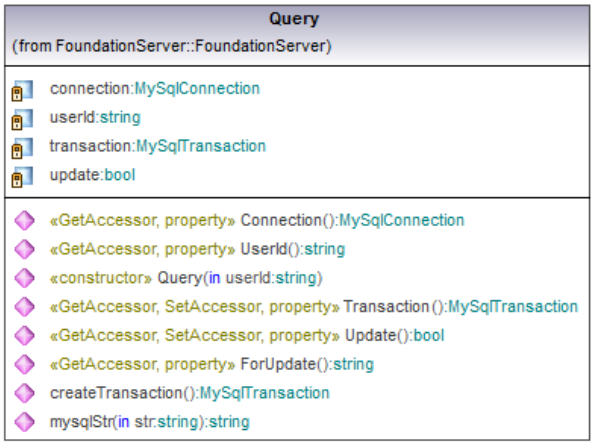


Ilustración 38: Clase Query del servidor

La clase define además métodos y elementos necesarios para solucionar la concurrencia y consistencia de datos que puede incumplir la lectura de información obsoleta (**ForUpdate**, **createTransaction**).

5.3.2.4. GENERACIÓN DINÁMICA DE ELEMENTOS

Debido a que algunos de los elementos persistentes requieren ser generados dinámicamente, el servidor ofrece una jerarquía y estructura que permiten definir algoritmos específicos para dicho propósito. La clase **Map** ofrece la estructura que necesita ser implementada para definir un tipo de generación concreta.

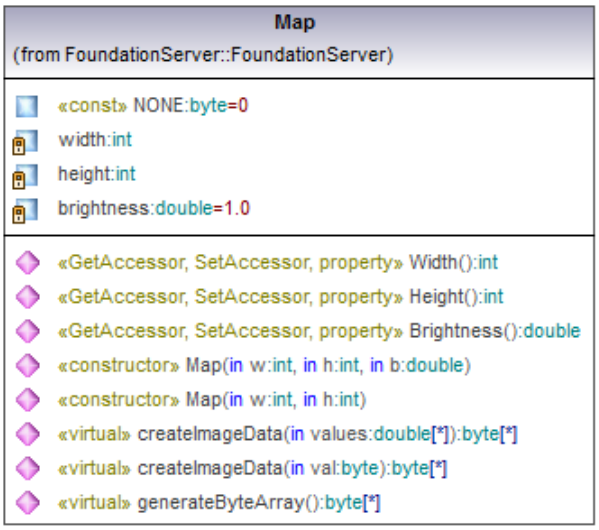


Ilustración 39: Clase Map del servidor

Mediante el método **generateByteArray**, las clases herederas que lo implementen definirán las operaciones y algoritmos que sean necesarios para generar un tipo de elemento. Es preciso indicar que la clase únicamente contempla la generación de elementos bidimensionales que estarán compuestos por matrices de bytes.

5.3.2.5. DEMONIOS DEL SERVIDOR

La aplicación del servidor requiere el uso de procesos que deben ser ejecutados periódicamente para cumplir con ciertos requisitos de la lógica del juego. Este tipo de procesos son implementados heredando de la clase **Batch**, la cual ofrece la estructura requerida para satisfacer dicha necesidad.

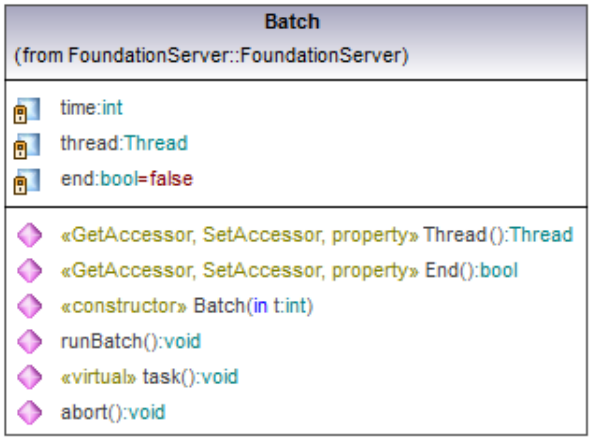


Ilustración 40: Clase Batch del servidor

Haciendo uso de procesos ligeros mediante la clase **Thread**, los objetos que hereden de la clase podrán definir mediante el método **task** las funciones que deben desempeñar de forma periódica. Los procesos se dormirán el tiempo especificado y se ejecutarán hasta que la aplicación servidor se detenga o hasta que se indique de forma implícita por medio del método **End**.

5.3.2.6. CONTROLADORES HTTP

Las peticiones que realiza el cliente al servidor son gestionadas mediante controladores HTTP, el comportamiento de estos controladores puede definirse heredando de la clase **IHTTPHandler** ofrecida por el framework ASP.NET. La lógica del juego y de las peticiones queda recogida en estos controladores haciendo uso de consultas y elementos persistentes (herederas de **Query** y **Persistent**).

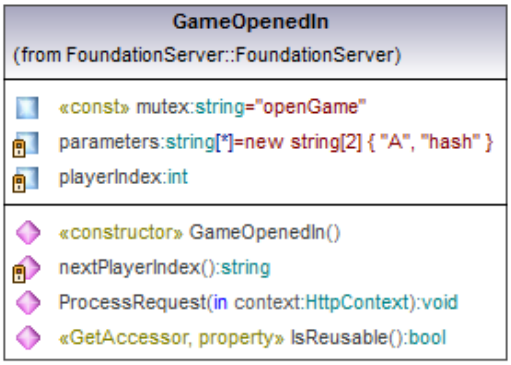


Ilustración 41: Clase GameOpenedIn del servidor

Todas las peticiones del cliente tienen un controlador asociado que hace uso del método **ProcessRequest** para enviar una respuesta válida. Las peticiones que lleguen al controlador deben de ser previamente comprobadas para evitar la redirección de peticiones a controladores erróneos.

5.4. DIAGRAMAS DE LAS BASE DE DATOS

El diseño de la base de datos de la aplicación, requiere de la identificación de los principales elementos que podrían estar presentes en el universo de juego, y al mismo tiempo, la identificación de los principales elementos y campos necesarios para llevar a cabo la representación tridimensional del mismo.

El universo contemplado en el juego, estará formado por sistemas y agujeros negros ubicados a lo largo de la extensión del mismo. A si mismo los sistemas que lo conforman, podrán contener o no planetas y estrellas, aunque todos ellos dispongan de campos de meteoritos. Los planetas por su parte, estarán compuestos por territorios y lunas que proporcionarán distintos tipos de recursos. Estos pueden ser conquistados por los jugadores con el fin de construir en ellos edificaciones que les permitan extraer sus recursos, crear naves de batalla o mejorar el ataque y defensa de estas últimas.

Las naves de batalla serán controladas por los jugadores y podrán moverse por el universo, atacar a otras unidades y conquistar territorios y lunas.

5.4.1. MODELO ENTIDAD-RELACIÓN

El modelo entidad-relación es una herramienta para el modelado de la información de una base de datos. Con él se puede indicar cuáles son las entidades relevantes de un sistema de información, sus atributos y observar físicamente sus relaciones. A continuación, se va a mostrar el modelo entidad-relación simplificado por cuestiones de espacio. En el que únicamente se muestran los identificadores de las entidades.

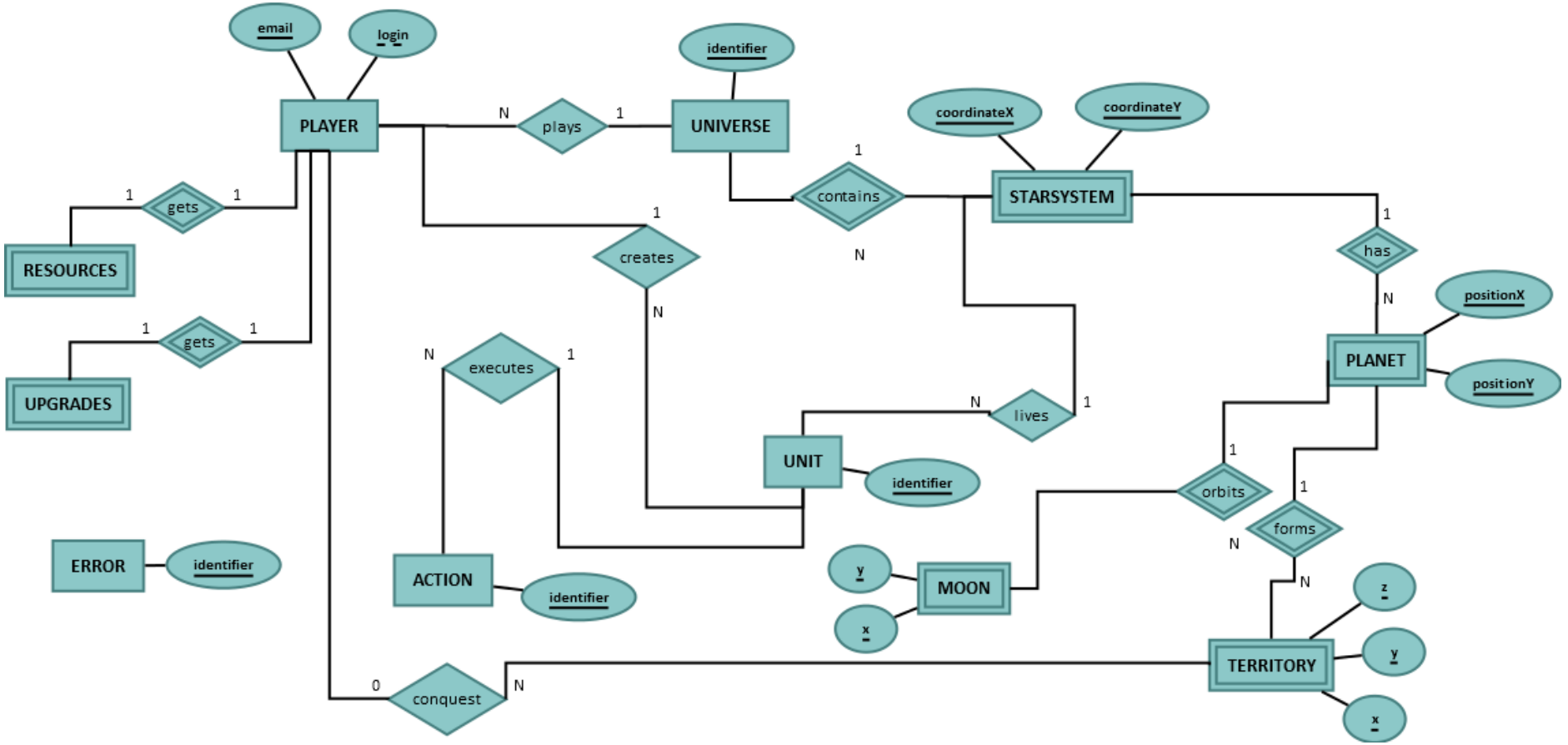


Ilustración 42: Modelo entidad-relación de la base de dato

Se han distinguido once tipos de entidades para almacenar de forma consistente el estado del juego:

- **PLAYER:** Destinada a almacenar la información personal del jugador. Su clave principal es el email del jugador.
- **RESOURCES:** Recoge la cantidad de recursos de los que dispone un determinado jugador. Se trata de una entidad débil puesto que su existencia no tiene sentido sin la del jugador correspondiente. Su clave principal es el nombre del jugador asociado.
- **UPGRADES:** Almacena el nivel de las mejoras de un determinado jugador. Se trata de una entidad débil puesto que su existencia no tiene sentido sin la del jugador correspondiente. Su clave principal es el nombre del jugador asociado.
- **UNIVERSE:** Destinada a almacenar información de los diversos universos generados por el juego. Dispone de un identificador único como clave principal.
- **STARSYSTEM:** Entidad débil que almacena información de los sistemas pertenecientes a un universo determinado. Su clave principal está compuesta por el correspondiente universo y sus coordenadas dentro del mismo.
- **PLANET:** Entidad débil destinada a almacenar información acerca de los planetas de un determinado sistema. Su clave está formada por un determinado sistema y su posición dentro del mismo.
- **TERRITORY:** Entidad débil que almacena información acerca de los territorios de un planeta determinado. Su clave está compuesta por un planeta y las coordenadas relativas al centro de este último.
- **MOON:** Entidad débil destinada a recoger información de las lunas que orbitan en torno a un planeta determinado. Su clave está compuesta por un planeta y las coordenadas relativas a la órbita de este último.
- **UNIT:** Almacena información acerca de las naves de batalla de las que disponen los jugadores. Dispone de un identificador único y secuencial como clave principal.
- **ACTION:** Recoge información acerca de las acciones llevadas a cabo por una determinada unidad. Su clave principal es un identificador único y secuencial.
- **ERROR:** Entidad que almacena información acerca de los errores de la aplicación servidor a modo de log. Los errores tienen un identificador único y secuencial como clave principal.

Las relaciones entre las entidades anteriormente descritas son las siguientes:

- **PLAYER-RESOURCES (gets):** Relación 1-1 e identificativa. Un jugador dispone únicamente de una cantidad de recursos determinada.
- **PLAYER-UPGRADES (gets):** Relación 1-1 e identificativa. Un jugador dispone únicamente de un nivel de mejoras determinado.
- **PLAYER-UNIVERSE (plays):** Relación N-1. En un universo juegan varios jugadores, aunque un jugador puede jugar en un único universo.
- **UNIVERSE-STARSYSTEM (contains)**: Relación 1-N e identificativa. Un universo dispone de varios sistemas, mientras que un sistema pertenece a un único universo.
- **STARSYSTEM-PLANET (has):** Relación 1-N e identificativa. Un universo tiene o no varios planetas, aunque un planeta pertenece a un único sistema.
- **PLANET-TERRITORY (forms):** Relación 1-N e identificativa. Un planeta está formado por varios planetas, mientras un territorio forma únicamente un planeta.
- **PLANET-MOON (has):** Relación 1-N e identificativa. Un planeta mantiene en su órbita o no varias lunas, aunque una luna puede orbitar en torno a un único planeta.
- **PLAYER-UNIT (creates):** Relación 1-N. Un jugador puede crear y controlar de cero a varias naves de batalla, mientras que una nave de batalla puede ser controlada y creada por un único jugador.

- **UNIT-ACTION (executes):** Relación 1-N. Una nave de batalla puede realizar o no acciones, mientras que una acción es realizada por una única nave de batalla.
- **PLAYER-TERRITORY (conquest):** Relación 0-N. Un jugador puede conquistar de cero a varios territorios, aunque un territorio puede estar conquistado por un único jugador.

5.4.2. MODELO LÓGICO DE LA BASE DE DATOS

Una vez realizado el modelo conceptual o de entidad-relación de una base de datos, es necesario llevar a cabo el diseño de un modelo lógico que pueda traducirse en última instancia en el modelo físico a implementar.

Este modelo representa las tablas reales que se introducirán en la base de datos, con sus diversos campos, que como se verá, algunas ya se recogían de una u otra forma en el modelo entidad-relación anteriormente observado, sólo que con distinto detalle. Las relaciones entre tablas son todas del tipo UC/DC (Actualización y borrado en cascada), a excepción de la existente entre **PLAYER** y **TERRITORY** (**conquest**) que es del tipo UC/DR (Actualización en cascada y borrado restringido) ya que no se debe eliminar el territorio si el jugador que lo conquistó desaparece.

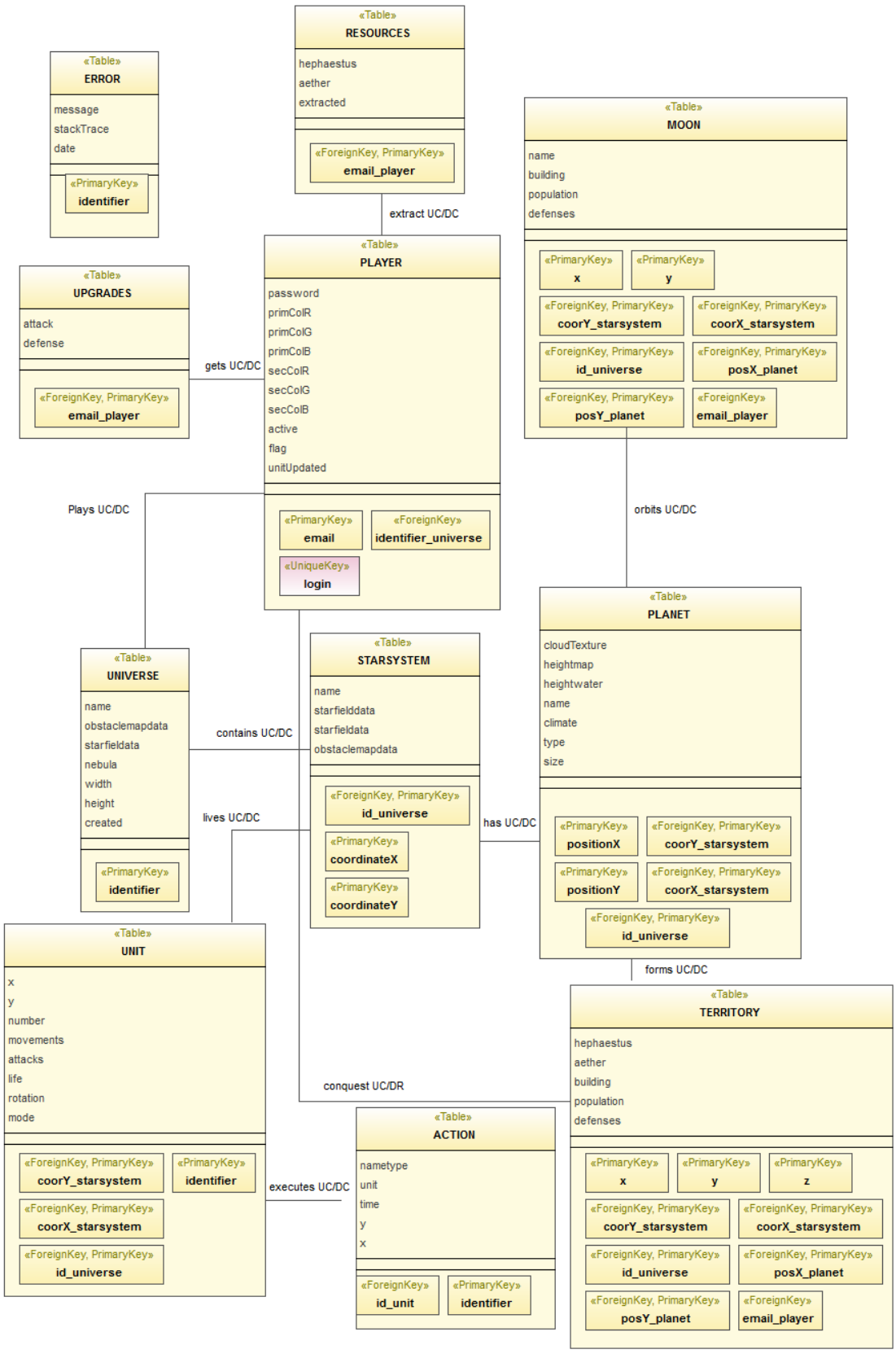


Ilustración 43: Diagrama lógico de la base de datos

6. IMPLEMENTACIÓN

Este capítulo describe en profundidad los detalles de implementación de los aspectos más relevantes de la aplicación, estructurados en apartados diferenciados y distinguiendo entre cliente y servidor.

6.1. CLIENTE

6.1.1. INTERACCIÓN CON ELEMENTOS TRIDIMENSIONALES

La interacción con los elementos que son representados en los diversos escenarios de juego se realiza mediante proyección.

La cámara del juego recoge la representación de aquellos elementos que se encuentren contenidos en una pirámide cuyo vértice, coincide con la posición de la cámara y cuya base, es un plano (plano distante) ubicado a una determinada distancia de la misma (horizonte). Además el campo de visión puede reducirse por un segundo plano paralelo al anterior (plano próximo), formando así un hexaedro irregular.

Por otra parte la interacción con el ratón ofrece únicamente coordenadas bidimensionales sobre un plano que se corresponde con la pantalla de nuestro ordenador personal y que coincide con el plano próximo de la cámara. Debido a esto, es necesario transformar las coordenadas tridimensionales de los elementos representados en coordenadas bidimensionales sobre la pantalla. Esta función se denomina proyección y es ofrecida por la librería XNA, únicamente requiere las coordenadas del objeto, la matriz proyección de la cámara, la matriz vista y la matriz mundo (las tres pertenecientes a la cámara) para calcular la posición bidimensional del objeto.

```
Vector3 posAux = this.Position;

Vector3 vpaux = base.Game.GraphicsDevice.Viewport.Project(posAux, this.Projection,
this.View, Matrix.Identity);

Vector2 vpaux2 = new Vector2(vpaux.X, vpaux.Y);
```

La posición que proyectaremos del objeto será su centro, aunque no basta con proyectar un único punto sobre la pantalla, ya que no sería usable para el jugador tener que ubicar el ratón en una posición tan precisa y a priori desconocida. Es por esto por lo que se permite un margen de error o tolerancia que puede interpretarse como una distancia al punto proyectado. Conceptualmente puede entenderse como una cubo que rodea el volumen del elemento y que proyectado sobre la pantalla formaría una cuadrado, en cuya área podríamos pulsar con el ratón para interactuar con dicho elemento.

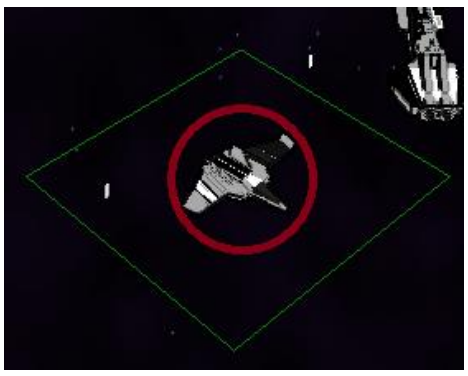


Ilustración 44: Nave de batalla y su área de selección (en rojo)

6.1.2. SISTEMA DE PARTÍCULAS

La animación de los objetos tridimensionales se ha implementado mediante un sistema de partículas bidimensional. Un sistema de partículas intenta representar el comportamiento de un conjunto de cuerpos que se encuentran bajo los efectos de fuerzas u otros elementos físicos. Puede considerarse como un pequeño y primitivo motor de física que se utiliza con una finalidad muy concreta y limitada.

De esta forma, efectos tales como explosiones, disparos laser y trayectorias siguen la lógica del sistema de partículas implementado. Por razones de sencillez, la implementación tiene en cuenta únicamente las leyes físicas de la velocidad y aceleración para determinar la posición de las partículas en un espacio bidimensional. Por tanto, la definición de partícula constará únicamente de posición, rotación, velocidad inicial, velocidad final y aceleración, además de una imagen que se utilizará como su representación en pantalla.

Con una inicialización previa de las partículas en función del efecto deseado e iterando sobre cada una de ellas, obtenemos un comportamiento realista que servirá para animar las diversas acciones de las naves de batalla incluidas en el juego.

```
foreach (Particle p in particles)
{
        if (p.Active)
        {
                Velocity += Acceleration * dt;
                Position += Velocity * dt;

                Rotation += RotationSpeed * dt;
                }
        }
}
```

Actualmente se representan tres efectos haciendo uso del sistema de partículas:

- **Movimiento:** Las naves de batalla dejan marcada su trayectoria en sus movimientos por los sistemas del universo. Las partículas de la trayectoria carecen de velocidad inicial y aceleración y hacen aparición durante el movimiento de la nave.
- **Ataque:** Las naves de batalla atacan haciendo uso del laser que se comporta como un grupo de partículas con velocidad inicial determinada y sin aceleración.
- **Explosión:** Las naves de batalla desaparecen del escenario tras una explosión provocada por el ataque de otras naves. Las explosiones siguen la lógica de partículas que tienen poca velocidad inicial y aceleración.

Ilustración 45: Trayectoria de movimiento con sistema de partículas

6.1.3. CULLING

Los algoritmos de culling buscan eliminar de la escena u ocultar todos aquellos elementos tridimensionales contenidos dentro del horizonte de la cámara que no son visibles desde una perspectiva dada. Esto permite ahorrar el cálculo computacional y conseguir una mejor tasa de FPS, sin que aparentemente nada haya cambiado en la imagen mostrada.

Para llevar a cabo esta tarea, los algoritmos de culling deben ejecutarse en bucle de forma continua mientras la aplicación está en funcionamiento. Esto puede suponer sin embargo un problema, ya que si el algoritmo de culling es demasiado extenso o costoso puede provocar justo el problema que estaba tratando de evitar: reducir la tasa de FPS.

Por ello, la lógica de un algoritmo de culling debe buscar ser lo más simple posible para no afectar la ejecución del bucle de la aplicación, pero al mismo tiempo debe producir resultados efectivos en la tasa de FPS.

La implementación escogida incluye un algoritmo básico de culling en la aplicación cliente, denominado Frustrum Culling. El algoritmo busca aquellos elementos que no colisionan con el hexaedro que conforma la vista de la cámara para no representarlos. De esta forma se ahorra representar elementos innecesarios sin que el usuario perciba un cambio en su visión del escenario.

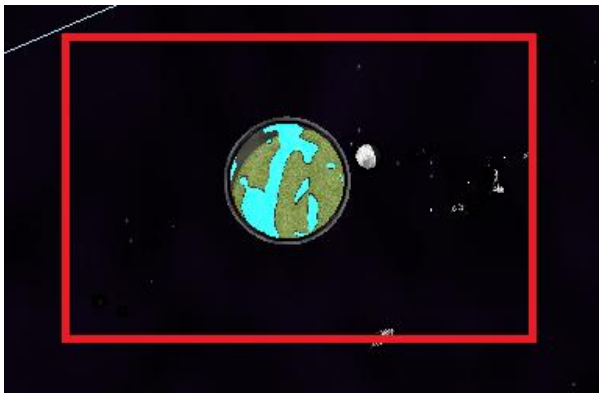


Ilustración 46: Representación del plano distante de la cámara

Para simplificar los cálculos requeridos, la colisión no se calculará con el cuerpo de los elementos representados, ya que esta operación puede resultar costosa si los elementos disponen de un gran número de vértices. Por ello la colisión se realizará con las esferas mínimas que rodean el volumen completo de los elementos y cuyo centro se ubica en la posición central de los mismos, ya que la colisión con una circunferencia es una operación simple que además es ofrecida por la librería XNA.

```
foreach (ModelMesh mesh in this.Model.Meshes)
{
        If(this.CompoundElement.ViewFrustrum.Intersects(mesh.BoundingSphere.Transform(
World)))
        {
                foreach (Effect effect in mesh.Effects)
                {
                        mesh.Draw();
                }
        }
}
```

6.2. SERVIDOR

6.2.1. POOL DE CONEXIONES

Debido a la naturaleza de la aplicación, el servidor debe mantener abiertas tantas conexiones a la base de datos como usuarios tenga conectados en un determinado instante de tiempo. Debido a que el número de usuarios estimado es alto y variable a lo largo del tiempo, no es viable abrir y cerrar conexiones bajo demanda de los jugadores (al abrir y cerrar la aplicación cliente), ya que se tratan de operaciones pesadas y de alto consumo de recursos.

Por este motivo es necesario crear un pool de conexiones [26] mediante el cual se mantenga abierto ilimitadamente un número fijo de conexiones. De esta forma se consigue evitar el problema del uso de recursos, aunque se requiere de lógica adicional para gestionar dichas conexiones.

Al ejecutar la aplicación servidor se inicia una lista de conexiones de tamaño constante (el tamaño óptimo de la lista puede calcularse fijando un máximo teórico de jugadores, de latencia y realizando pruebas de estrés) que no se cerrarán hasta que la aplicación se detenga. Cuando un jugador se conecta es preciso asignarle una conexión de la lista para que haga uso de ella, preferiblemente que no esté ya en uso. En caso de no existir conexiones sin uso, es necesario indicar una política mediante la que se van liberando conexiones.

```
if (playersConnections.ContainsKey(player))
{
        con = playersConnections[player];
}
else
{
        if (playersConnections.Count >= MAXCONNECTIONS)
        {
                playersConnections.Add(player, cons[0]);
                con = cons[0];
        }
        else
        {
                int index = playersConnections.Count;
                playersConnections.Add(player, connections[index]);
                con = connections[index];
        }
}
```

La política implementada para el pool de la aplicación es **FIFO**, de forma que el primer usuario al que se le asignó una conexión, será el primero en perderla para que sea asignada a un nuevo jugador. En caso de que el número de jugadores sea mayor que el de la lista de conexiones, los jugadores tendrán que esperar a que el jugador con más tiempo en posesión de la conexión deje de usarla para que pueda ser asignada a uno nuevo (aumentando así la latencia entre cliente y servidor).

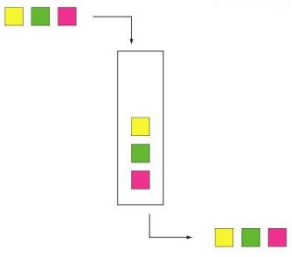


Ilustración 47: Política FIFO

6.2.2. CONCURRENCIA DE DATOS

La naturaleza masiva del juego provoca una interacción múltiple de los jugadores sobre elementos comunes, tales como las naves de batalla, los planetas, los territorios, etc. Esto ocasiona problemas de concurrencia sobre los elementos almacenados en la base de datos, ya que al atender peticiones de forma paralela, es posible que un jugador realice cambios sobre elementos que han sido actualizados por otro jugador (lectura obsoleta), poniendo de esta forma en riesgo la consistencia de los datos y del estado del juego.

La solución implementada hace uso de bloqueos para garantizar la consistencia de la información almacenada en la base de datos. Aunque esta técnica supone incluir retardos en la comunicación debido a la espera por la liberación del recurso bloqueado, es la técnica de mayor sencillez y más utilizada ante un problema de este tipo.

Las técnicas y elementos de bloqueo clásicos (semáforos y variables condicionales) quedan descartados debido a la gran cantidad de elementos de control necesarios de los que harían uso los elementos de la base de datos (un semáforo por cada fila comprometida de la base de datos). Además dicho número es indeterminado, debido a que no es posible conocer a priori el número de jugadores que harán uso de un número de elementos que también es desconocido. Es por tanto necesario recurrir a las técnicas de bloqueo ofrecidas por la base de datos.

Mediante transacciones, las bases de datos definen contextos en los que realizar acciones (consultas, inserciones, modificaciones y borrados) de forma segura y consistente. Se ofrecen también mecanismos que permiten anular durante la ejecución las acciones realizadas, en caso de que nuestra lógica de aplicación lo requiera (operaciones de **commit** y **rollback**).

De entre los elementos ofrecidos por MySQL para llevar a cabo el bloqueo de filas, se ha utilizado la sentencia bloqueante SELECT…FOR UPDATE [27]. Esta sentencia, realizada en un contexto de transacción, permite bloquear un elemento que ha sido consultado hasta su actualización o hasta el fin de la transacción. El resto de jugadores que accedan al elemento permanecerán a la espera hasta que sea liberado, momento en el que el siguiente de ellos bloqueará dicho elemento (formando así una cola de espera para los elementos bloqueados).

```
string qSql = "SELECT building,population,defenses FROM territory WHERE
player=" + mysqlStr(this.player.Login) + base.ForUpdate + ";";

MySqlCommand qry = new MySqlCommand(qSql, base.Connection);
qry.Transaction = base.Transaction;
MySqlDataReader qReader = qry.ExecuteReader();
```

6.2.3. CRECIMIENTO Y GENERACIÓN DINÁMICA DEL UNIVERSO

Debido al gran, variable y desconocido número de jugadores que hará uso de la aplicación, no es viable la generación previa de los universos por medio de herramientas de modelado, es requisito por tanto la generación dinámica del universo y los elementos que lo componen.

UNIVERSO

El universo está compuesto visualmente por un fondo de estrellas y nebulosas características. Ambas son generadas y almacenadas por el servidor como texturas que se envían al cliente bajo petición y generación hace uso del algoritmo de Perlin con salida bidimensional. La representación de la salida de

Perlin queda en manos de la aplicación cliente, aunque la función de ruido usada garantiza la distinción visual de los universos.

Ilustración 48: Nébula y campo de estrellas de universo

Mientras que los agujeros negros que componen el universo son ubicados en el espacio por medio de la función de ruido utilizada anteriormente, los sistemas que lo componen utilizan un algoritmo más concreto. Bajo demanda de los jugadores que se registran en el juego, el servidor construye y ubica los sistemas en torno a la posición central del universo. Si un usuario es registrado en la aplicación, se le asigna el planeta libre del sistema más próximo al centro del universo. En caso de no existir ningún planeta con estas características, el servidor genera un conjunto de sistemas alrededor de los ya creados (a modo de anillo) sobre los que reanudará la búsqueda para asignar al jugador un planeta disponible.

Ilustración 49: Primer y segundo anillos de sistemas del universo

SISTEMAS

Los sistemas disponen de un campo de estrellas y de una nebulosa que es generado de forma idéntica al de los universos. Los sistemas disponen de dos tipos de elementos generados: Planetas y campos de asteroides. Los campos de asteroides utilizan la misma lógica que los agujeros negros del universo aunque dispongan de una representación y densidad distinta (Mayor número de asteroides).

```
byte[] dataB = null;
List<PerlinNoise2D> noiseFunctions = new List<PerlinNoise2D>();
List<double> summedValues;

if (this.type == NEBULA)
{
        noiseFunctions.Add(new PerlinNoise2D(16, 20f));
        noiseFunctions.Add(new PerlinNoise2D(8, 10f));
        noiseFunctions.Add(new PerlinNoise2D(4, 5f));

        summedValues = sumNoiseFunctions(base.Width, base.Height, noiseFunctions);
        dataB = createImageData(summedValues.ToArray());
}
```

Por otra parte, los planetas se ubican en torno a la posición central del sistema, donde es posible encontrar una estrella. La ubicación se calcula aleatoriamente sobre anillos con centro la estrella, el radio dependerá del número de planetas que se vaya crear y aumenta tras ubicar un planeta.

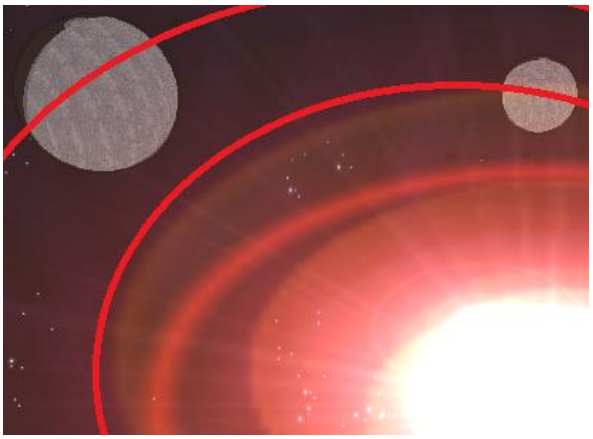


Ilustración 50: Primer y segundo anillo de planetas del sistema

PLANETAS

Los planetas están compuestos e identificados visualmente por una atmósfera, océanos y continentes. La generación de la atmósfera se realiza en el servidor por medio de un algoritmo fractal (fractal plasma) de una forma similar a la generación de la nébula de los sistemas mediante ruido.

Los océanos y continentes se generan con una única textura generada mediante ruido de Perlin. Únicamente se fija un valor umbral sobre la salida de la función por debajo del cual se representa agua y por encima del mismo terreno. De esta forma se consigue el aspecto deseado de los planetas que variarán la textura del terreno en función de su proximidad o lejanía a la estrella central.

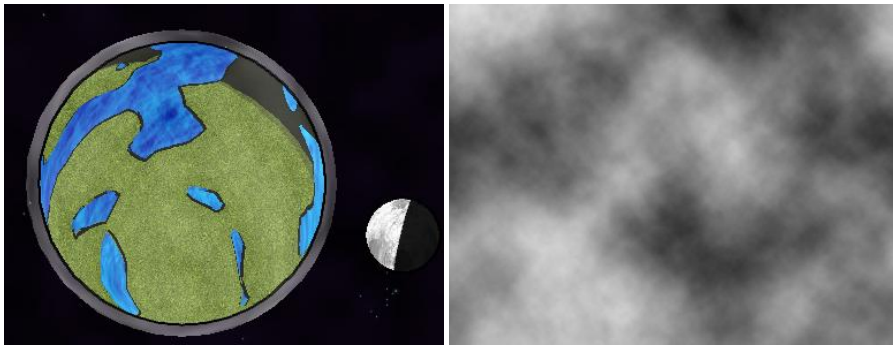


Ilustración 51: Atmósfera, océano y continentes del planeta y textura de ruido

6.2.4. BASE DE DATOS

La base de datos de la aplicación servidor ha sido implementada en una máquina virtual con sistema operativo Windows Server 2003, usando VMware Server. Para ello, se ha instalado el sistema gestor de bases de datos MySQL Server 5.5, versión gratuita y descargable desde su página web.

Una vez finalizada la instalación, se ha procedido a crear diversos scripts SQL que permiten implementar el modelo lógico mostrado en el apartado **5.4.2**.

- Script de creación de tablas y relaciones.
- Script de creación de roles de la base de datos (Rol administrador y rol usuario).
- Script de creación de usuarios que implementan los roles creados.
- Script de creación y asignación de permisos a los usuarios creados.

7. CONCLUSIONES Y FUTUROS DESARROLLOS

Este capítulo recoge las conclusiones extraídas en la fase final de desarrollo del proyecto y una lista de posibles ideas para ampliarlo en futuros desarrollos.

7.1. CONCLUSIONES

El proyecto por parte de la aplicación cliente, ha permitido adquirir una gran cantidad y diversidad de conocimientos acerca del desarrollo de aplicaciones con interfaz tridimensional. Se han podido comprender en detalle aspectos como la lógica interna que controla el funcionamiento de una aplicación de este tipo, la representación tridimensional resultante de la traducción entre espacios de coordenadas, la implementación de efectos mediante shaders, los algoritmos de optimización propios del renderizado, los mecanismos de interacción con elementos tridimensionales, la generación dinámica de elementos gráficos mediante algoritmos de modelado procedural, etc.

Todos estos conocimientos no están limitados al uso de la librería XNA, si no que sus conceptos, aunque varíen su forma con cada implementación, son compartidos por todos los motores de desarrollo de videojuegos.

Por parte de la aplicación servidor, se han adquirido gran variedad de conceptos acerca del desarrollo e implementación de servidores concurrentes, aunque muchos conceptos eran conocidos desde el punto de vista teórico, poder afrontar una implantación ha hecho que fuese necesario ahondar en muchos de ellos. Entre estos conceptos se encuentra la concurrencia de las bases de datos, la gestión de conexiones en un servidor con elevado número de clientes, la seguridad en las comunicaciones atendiendo a las necesidades de integridad, autenticación y confidencialidad, la gestión de demonios y trabajos por lotes, etc.

El entorno de ejecución ha sido otra gran fuente de conocimientos, ya que gran parte de la problemática de implementación surgió de una configuración errónea del servidor de aplicaciones y del despliegue de la aplicación en el servidor. Enfrentarse a los problemas encontrados y solucionarlos ha sido una experiencia que ha aportado grandes conocimientos en el ámbito de las arquitecturas cliente-servidor

Respecto a la aplicación en conjunto e infraestructura común, ha podido comprobarse la gran relevancia de cada una de las decisiones y elecciones escogidas y la gran dependencia del diseño de estas últimas. Aspectos como el protocolo de red utilizado o la arquitectura de red escogida para sus nodos, hacen variar casi en su totalidad el diseño y funcionamiento de la aplicación.

Cabe destacar la poca documentación existente acerca de la problemática multijugador masiva, que agrava aún más la complejidad del proyecto desarrollado, puesto que aplicaciones de este tipo requieren para su diseño e implementación de grupos de trabajo que cuentan con números recursos humanos y altos conocimientos en la materia (expertos en comunicaciones, diseñadores de entornos tridimensionales, etc.).

En definitiva, este proyecto ha permitido llevar a cabo el principal objetivo del mismo: Estudiar la problemática de los juegos multijugador masivos y diseñar e implementar un juego de estrategia con dichas características (MMORTS). Además, desde un punto de vista objetivo, ha permitido conocer en profundidad un campo de interés personal y que no es explorado durante la carrera.

Aunque finalmente el producto desarrollado no ha sido comercializado debido al alto coste de mantenimiento que hubiese supuesto, ha sido lucrativo investigar los diversos servicios de publicación tales como Steam o Xbox Live Arcade.

7.2. LÍNEAS FUTURAS

A continuación se detalla una serie de posibles ampliaciones que pueden continuarse en el proyecto presentado para convertirlo en un juego más completo y que por cuestiones de coste y tiempo, no han sido incluidas.

- **Animaciones:** Las animaciones mostradas en la aplicación son visualmente pobres y de tipo básico (translaciones y rotaciones) y no hacen uso de técnicas de animación de modelos tridimensionales más complejas como el uso de esqueletos [28].
- **Análisis de rendimiento del servidor:** El servidor de la aplicación no ha sido sometido a pruebas de estrés en el que un gran número de jugadores hacen uso de la aplicación. No se ha calculado por tanto el número aceptable de conexiones para un máximo determinado de jugadores y latencia respecto al servidor.
- **Occlusion culling:** El algoritmo de optimización del renderizado implementado es el más sencillo y común de entre los utilizados. Aunque se estudió el funcionamiento del algoritmo de occlusion culling [29] y se analizó su viabilidad y mejora en rendimiento, no llegó a ser implementado.
- **Aumento de la jugabilidad:** Incluyendo nuevas unidades capaces de realizar nuevas acciones, nuevas edificaciones o nuevos recursos.
- **Comunicación entre jugadores:** Los jugadores de la aplicación se encuentran incomunicados unos de otros debido a que la aplicación no ofrece ningún sistema de comunicación. En pro de la jugabilidad sería conveniente desarrollar un sistema de mensajería que permita a los jugadores registrados enviar y recibir mensajes.
- **Artefactos en modelos:** El uso de modelos gratuitos para las naves de batalla, ha ocasionado errores visuales a la hora de representarlas en pantalla. Es conveniente corregir el modelado o hacer uso de modelos propios.
- **Seguridad:** La mejor solución de seguridad ante una arquitectura como la diseñada para la aplicación, es hacer uso de una PKI [30] y usar el protocolo de nivel de aplicación HTTPS. Este tipo de arquitectura soluciona las vulnerabilidades de Man-In-The-Middle [31] de las aplicaciones que hacen uso de ella, además de garantizar la integridad, confidencialidad y autenticación de la información transmitida.
- **Optimización de la base de datos:** Es posible obtener un mayor rendimiento en la persistencia de la información si se realiza un análisis estadístico previo de los datos almacenados y se configuran debidamente el tipo de almacenamiento, el tamaño de bloque. se indizan tablas teniendo en cuenta las consultas más frecuentes, etc.
- **Sonido:** La actual versión del juego carece de efectos sonoros que podrían ser fácilmente introducidos para incrementar la calidad de juego.
- **Configuración:** El juego carece de ningún tipo de configuración por parte del jugador, por lo que sería deseable implementar un estado de juego que permita desempeñar dicha tarea. Aspectos gráficos o de control deberían ser configurables desde la interfaz de juego.

8. PLANIFICACIÓN Y PRESUPUESTO

Este apartado recoge una descripción detallada de la distribución de tiempo dedicado al desarrollo del proyecto, el cual está dividido en varias fases, así como el presupuesto asociado al mismo.

8.1. PLANIFICACIÓN

La planificación del proyecto contempla varias etapas con inicio y fin en el tiempo. Cada una de estas etapas define una parte del desarrollo con un objetivo concreto, debidamente delimitada y diferenciada del resto. Las fases del proyecto y sus objetivos son los siguientes:

- **1ª Fase:** Estudio del mercado actual de juegos multijugador masivos de estrategia y abstracción de sus características comunes.
- **2ª Fase:** Estudio de la representación tridimensional y de las principales técnicas de modelado.
- **3ª Fase:** Estudio del modelado procedural y de funciones fractales y de ruido asociadas.
- **4ª Fase:** Estudio de los motores de videojuegos y bibliotecas de desarrollo gráfico, y sus principales alternativas.
- **5ª Fase:** Elección de una librería gráfica sobre la que desarrollar el proyecto.
- **6ª Fase:** Estudio en profundidad, aprendizaje y documentación sobre el funcionamiento, características y pruebas de la librería XNA.
- **7ª Fase:** Estudio de las diversas arquitecturas de red, protocolos de aplicación y frameworks para el desarrollo de servidores.
- **8ª Fase:** Elección del framework ASP.NET y de la arquitectura cliente-servidor inherente al framework.
- **9ª Fase:** Estudio detallado, análisis de los diversos módulos y realización de pruebas sobre el framework ASP.NET.
- **10ª Fase:** Estudio de las alternativas gratuitas de bases de datos ofrecidas por el mercado.
- **11ª Fase:** Elección de MySQL, estudio de los mecanismos de concurrencia de datos ofrecidos por el sistema gestor de las bases de datos escogido y pruebas acerca de los mismos.
- **12ª Fase:** Análisis de la aplicación.
- **13ª Fase:** Diseño de la aplicación.
- **14ª Fase:** Implementación del cliente.
- **15º Fase:** Implementación del servidor.
- **16ª Fase:** Implementación de los componentes de comunicación de cliente y servidor.
- **17ª Fase:** Configuración del servidor y despliegue de la aplicación en el mismo.
- **18ª Fase:** Realización de pruebas y corrección de errores.
- **19ª Fase:** Realización de pruebas con múltiples jugadores y corrección de errores.
- **20ª Fase:** Revisión y verificación de todo el trabajo del proyecto, y creación de la presentación y diapositivas.

Las pruebas fueron realizadas a pesar de que el presente documento no incluya un capítulo de pruebas. Esta fase de pruebas se caracteriza por la realización de dos tipos de pruebas:

- **Pruebas unitarias:** Destinadas a comprobar que todas las funciones devolviesen valores correctos para todo tipo de parámetro de entrada.
- **Pruebas de aceptación:** Destinadas a verificar que la aplicación cumplía con todos los requisitos y funcionalidades definidas en fase de análisis.

La fase de documentación y desarrollo del presente documento no se encuentra definida en la lista de fases, esto se debe a que dicha tarea ha sido realizada de forma paralela a las diversas fases del proyecto.

La siguiente tabla recoge las fechas de inicio, fin y duración de cada una de las fases del proyecto. No se han incluido los fines de semana, aunque sí las festividades coincidentes con días laborables.

Fases	Fecha de inicio	Fecha de fin	Duración
1ª Fase	28/10/2010	16/11/2010	14 días
2ª Fase	17/11/2010	03/12/2010	13 días
3ª Fase	06/12/2010	27/12/2010	16 días
4ª Fase	28/12/2010	17/01/2011	15 días
5ª Fase	18/01/2011	27/01/2011	8 días
6ª Fase	28/01/2011	03/03/2011	25 días
7ª Fase	04/03/2011	23/03/2011	14 días
8ª Fase	24/03/2011	15/04/2011	17 días
9ª Fase	18/04/2011	16/05/2011	21 días
10ª Fase	17/05/2011	23/05/2011	5 días
11ª Fase	24/05/2011	10/06/2011	14 días
12ª Fase	13/06/2011	04/07/2011	16 días
13ª Fase	05/07/2011	29/07/2011	19 días
14ª Fase	01/08/2011	04/11/2011	70 días
15ª Fase	07/11/2011	24/02/2012	80 días
16ª Fase	25/02/2012	25/04/2012	43 días
17ª Fase	26/04/2012	27/06/2012	45 días
18ª Fase	28/06/2012	24/07/2012	19 días
19ª Fase	25/07/2012	21/08/2012	20 días
20ª Fase	22/08/2012	03/09/2012	9 días

Tabla 125: Fases de planificación del proyecto

Han sido necesarios 483 días durante los cuales se ha realizado una media de 4 horas diarias, sumando un total de 1932 horas. La planificación queda debidamente recogida en el siguiente diagrama de Gantt.

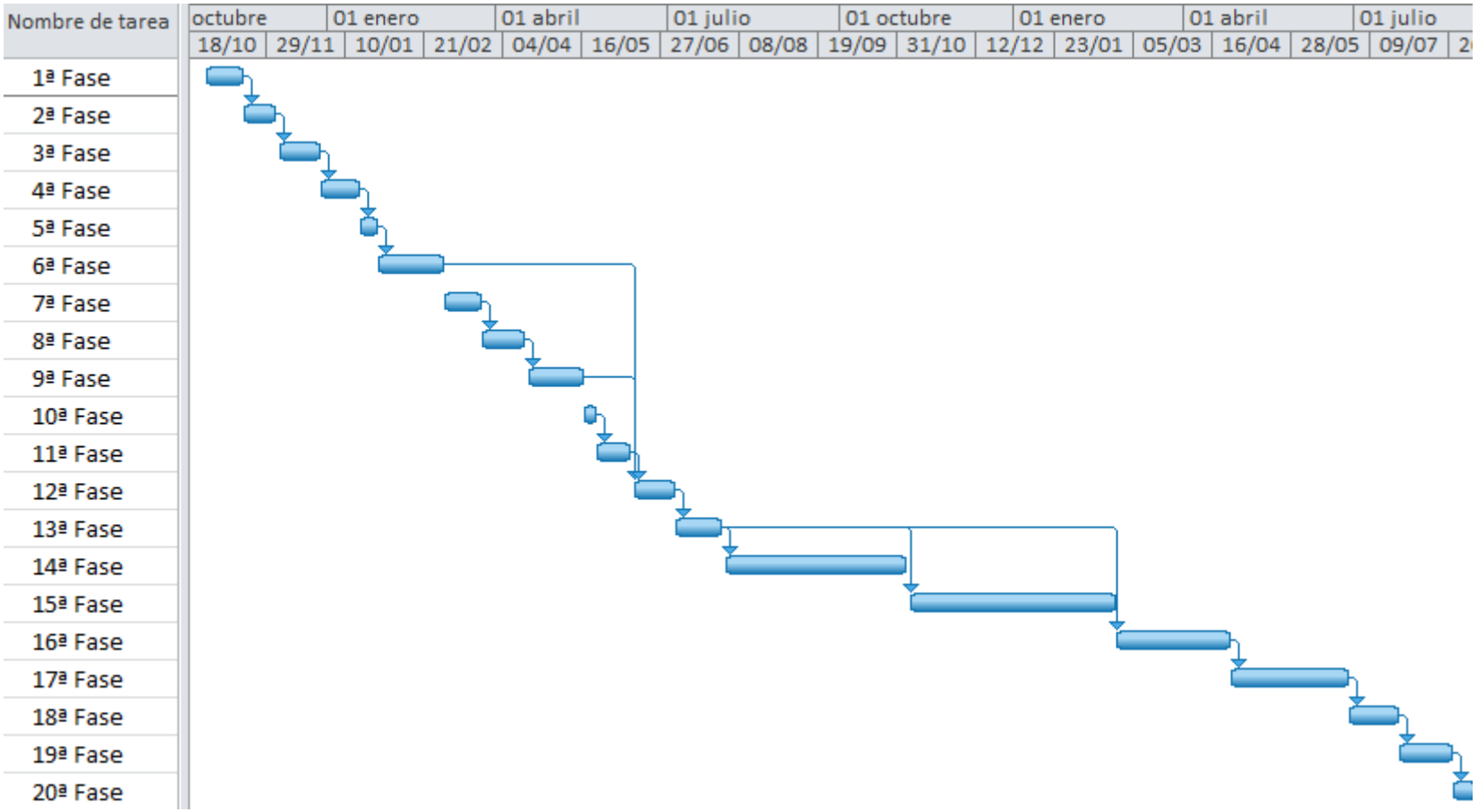


Ilustración 52: Diagrama de Gantt

8.2. PRESUPUESTO

En este punto se va a detallar el presupuesto del proyecto. Separados en distintos apartados se diferenciarán los costes de personal, hardware, software, material fungible, y costes indirectos, para finalmente calcular el coste total.

8.2.1. COSTE DE PERSONAL

El proyecto ha sido realizado en su totalidad por un único trabajador (el autor del mismo), asumiendo los múltiples roles involucrados en cada fase de trabajo. A continuación, se detalla el coste de cada una de estas fases indicando el rol y las horas dedicadas.

Fase	Rol	Coste(€/hora)	Horas	Coste total
1ª Fase	Analista	40,00	56	2240,00
2ª Fase	Analista	40,00	52	2080,00
3ª Fase	Analista	40,00	64	2560,00
4ª Fase	Analista	40,00	60	2400,00
5ª Fase	Analista	40,00	32	1280,00
6ª Fase	Analista	40,00	100	4000,00
7ª Fase	Analista	40,00	56	2240,00
8ª Fase	Analista	40,00	68	2720,00
9ª Fase	Analista	40,00	84	3360,00
10ª Fase	Analista	40,00	20	800,00
11ª Fase	Analista	40,00	56	2240,00
12ª Fase	Analista	40,00	64	2560,00
13ª Fase	Analista	40,00	76	3040,00
14ª Fase	Programador	25,00	280	7000,00
15ª Fase	Programador	25,00	320	8000,00
16ª Fase	Programador	25,00	172	4300,00
17ª Fase	Programador	25,00	180	4500,00
18ª Fase	Ingeniero de pruebas	30,00	76	2280,00
19ª Fase	Ingeniero de pruebas	25,00	80	2000,00
20ª Fase	Analista	40,00	36	1440,00
TOTAL			1932	61040

Ilustración 53: Coste de personal

8.2.2. COSTE DE HARDWARE

El desarrollo del proyecto ha requerido el uso de dos equipos: un ordenador de sobremesa y un servidor. Inicialmente se van a describir las características de ambos equipos para posteriormente detallar el coste de cada uno de ellos:

- **Ordenador de sobremesa:**
  - **Procesador:** Intel® Core™ i5 M430 @ 2.27 GHz.
  - **Memoria RAM:** 4 GB.
  - **Disco duro:** 500 GB ATA.
  - **Tarjeta de vídeo:** Ati Mobility Radeon HD 5470.
- **Servidor:**
  - **Procesador:** Intel® Xeon® CPU E5450 @ 3 GHz.
  - **Memoria RAM:** 8 GB.
  - **Disco duro:** 1.2 TB SCSI.

Equipo	Coste (€)	Dedicación (meses)	Uso dedicado al proyecto (%)	Periodo de depreciación	*Coste imputable (€)
Ordenador de sobremesa	800,00	23	100	60	306,67
Servidor	3000,00	3	100	60	150
TOTAL					456,67

Ilustración 54: Coste de hardware

***Fórmula de amortización:** $Coste\ imputable = \frac{A}{B} \cdot C \cdot D$

A: Meses de dedicación del equipo al proyecto.

B: Periodo de depreciación (60 meses).

C: Coste del equipo (sin IVA).

D: tanto por ciento del uso que se dedica al proyecto.

8.2.3. COSTE DE SOFTWARE

El ordenador de sobremesa y el servidor con los que se ha desarrollado el proyecto usan un sistema operativo Windows 7 Home Premium y un Windows Server 2003, adquiridos con licencia gratuita para estudiantes a través de la universidad. La aplicación Microsoft Project 2007 que ha sido utilizada para planificar el proyecto se ha adquirido también mediante este sistema. El resto del software ha sido adquirido de forma gratuita a través de internet.

La documentación del proyecto se ha realizado con Microsoft Office 2007 Hogar y Estudiante, la librería XNA ha sido la utilizada para desarrollar la aplicación cliente, VMware Server es el software utilizado para la creación de la máquina virtual que funciona como servidor, UltiDev Cassini es el servidor de aplicaciones sobre el que se despliega la aplicación servidor, MySQL Server 5.5 es el gestor de las bases de datos utilizado y Altova UModel Basic Edition es el programa con el que se han llevado a cabo los diagramas involucrados en el análisis y el diseño de la aplicación.

Software	Coste (€)
Licencia de Microsoft Windows 7 Home Premium	0,00
Licencia de Microsoft Office 2007 Hogar y Estudiantes	99,00
Licencia de Microsoft Office Project 2007	0,00
Licencia de XNA 4.0	0,00
Licencia de VMware Server	0,00
Licencia de MySQL Server 5.5	0,00
Licencia de Microsoft Windows Server 2003	0,00
Licencia de Altova UModel Basic Edition	119,00
Ultidev Cassini Web Server 2.0	0,00
TOTAL	218,00

Ilustración 55: Coste de software

8.2.4. MATERIAL FUNGIBLE

El coste asociado a consumibles (papel, CDs, etc.) es de 40 €.

8.2.5. COSTES INDIRECTOS

Los costes de las instalaciones donde se ha llevado a cabo el proyecto (luz, agua, internet, etc.) es de un 20% del coste total del proyecto (valor estimado).

8.2.6. COSTE TOTAL

Sumando los costes anteriormente calculados, se procede a calcular el coste total sin aplicar riesgo, IVA ni porcentaje de beneficio.

UNIVERSIDAD CARLOS III DE MADRID
Escuela Politécnica Superior

PRESUPUESTO DE PROYECTO

1.- Autor: Jose Luis Nieto García

2.- Departamento: Departamento de Informática

3.- Descripción del Proyecto:

- Título: **Creación de MMORTS**
- **Duración (meses):** **23**
- Tasa de costes Indirectos: 20%

4.- Presupuesto total del Proyecto (valores en Euros):

Euros

5.- Desglose presupuestario (costes directos)

PERSONAL

Apellidos y nombre	N.I.F. (no rellenar - solo a título informativo)	Categoría	Dedicación (hombres mes) *)	Coste hombre mes	Coste (Euro)	Firma de conformidad
Nieto García, Jose Luis		Analista	6,278095238	5.250,00	32.960,00	
Nieto García, Jose Luis		Programador	7,253333333	3.281,25	23.800,00	
Nieto García, Jose Luis		Ingeniero de pruebas	1,188571429	3.937,50	4.680,00	
					0,00	
					0,00	
		Hombres mes	**14,72**	**Total**	61.440,00	

*) 1 Hombre mes = 131,25 horas. Máximo anual de dedicación de 12 hombres mes (1575 horas)
Máximo anual para PDI de la Universidad Carlos III de Madrid de 8,8 hombres mes (1.155 horas)

EQUIPOS

Descripción	Coste (Euro)	% Uso dedicado proyecto	Dedicación (meses)	Periodo de depreciación	Coste imputable *)
Ordenador de sobremesa	800,00	100	23	60	306,67
Servidor	3.000,00	100	3	60	150,00
				Total	456,67

*) Fórmula de cálculo de la Amortización:

$$\frac{A}{B} x C x D$$

A = nº de meses desde la fecha de facturación en que el equipo es utilizado
B = periodo de depreciación (60 meses)
C = coste del equipo (sin IVA)
D = % del uso que se dedica al proyecto (habitualmente 100%)

SUBCONTRATACIÓN DE TAREAS

Descripción	Empresa	Coste imputable
	Total	0,00

OTROS COSTES DIRECTOS DEL PROYECTO*)

Descripción	Empresa	Costes imputable
Altova Umodel Basic Edition		119,00
Material fungible		40,00
Office 2007 Hogar y Estudiante		99,00
	Total	258,00

*) Este capítulo de gastos incluye todos los gastos no contemplados en los conceptos anteriores, por ejemplo: fungible, viajes y dietas, otros,...

6.- Resumen de costes

Presupuesto Costes Totales	Presupuesto Costes Totales
Personal	61.440
Amortización	457
Subcontratación de tareas	0
Costes de funcionamiento	258
Costes Indirectos	12.431
Total	74.586

Ilustración 56: Presupuesto calculado con la plantilla de rúbrica

9. GLOSARIO

- **MMORTS:** Multiplayer Massive Online Real Time Strategy (Videojuego de Estrategia en Eiempo Real Multijugador Masivo y en Línea).
- **MIT:** Massachusetts Institute of Tecnology (Instituto Tecnológico de Massachusetts).
- **BBS:** Bulletin Board System (Sistema de Tablón de Anuncios).
- **MUD:** Multi User Dungeon (Mazmorra Multiusuario).
- **FPS:** First Person Shooter (Videojuego de Disparos en Primera Persona).
- **PAN:** Personal Area Network (Área de Red Personal).
- **LAN:** Local Area Network (Área de Red Local).
- **MAN:** Metropolitan Area Network (Área de Red Metropolitana).
- **WAN:** Wide Area Network (Área de Ámplia).
- **MAC:** Media Access Control (Control de Acceso al Medio).
- **OSI:** Open System Interconnection (Modelo de Interconexión de Sistemas Abiertos).
- **Streaming:** Distribución de multimedia a través de una red de manera que el usuario consume el producto al mismo tiempo que se descarga.
- **DARPA:** Defense Advanced Research Projects Agency (Agencia de Investigación de Proyectos Avanzados de Defensa).
- **TFTP:** Trivial File Transfer Protocol (Protocolo de Transferencia de Archivos Trivial).
- **POP:** Post Office Protocol (Protocolo de Oficina de Correos).
- **Telnet:** Telecommunication Network (Red de Telecomunicación).
- **HTTPS:** Hyper Text Transfer Protocol Secure (Protocolo Seguro de Transferencia de Hypertexto).
- **SFTP:** Secure File Transfer Protocol (Protocolo de Transferencia Segura de Archivos).
- **MIME:** Multipurpose Internet Mail Extension (Extensiones Multipropósito de Correo de Internet).
- **Cookies:** Pequeña pieza de información enviada por un sitio web, las cual son almacenadas en el navegador del usuario.
- **URL:** Uniform Resource Locator (Localizador de Recursos Uniforme).
- **ASP:** Active Server Pages. Tecnología de Microsoft para generar páginas web dinámicas.
- **Visual Basic:** Lenguaje de programación de Microsoft orientado a eventos.
- **J#:** Lenguaje de programación basado en Java creado para desarrollar aplicaciones para la plataforma .NET.
- **XML:** Extensible Markup Languaje (Lenguaje de Marcas Extensible).
- **SGML:** Estándar Generalized Markup Languaje (Estándar de Lenguaje de Marcado Generalizado). Sistema orientado a la organización y etiquetado de documentos.
- **DTD:** Document Type Definition (Definición de Tipo de Documento).
- **Difusión:** Acuñado por Claude Shannon para referirse al efecto logrado por las permutaciones realizadas por un cifrador.
- **Confusión:** Acuñado por Claude Shannon para referirse al efecto logrado por las sustituciones realizadas por un cifrador.
- **Triple DES:** Algoritmo que realiza tripe cifrado DES, convido para aumentar la longitud de clave de DES sin cambiar el algoritmo.
- **S-Cajas:** Componente de los algoritmos de cifrado de clase simétrica para conceder confusión al texto cifrado.
- **Criptoanálisis diferencial:** Técnica para el análisis criptográfico que consiste en cifrar parejas de mensajes escogidos que obedecen a un patrón definido.

- **XOR:** Disyunción lógica de dos operandos también conocida como o exclusiva que es verdad si únicamente un operando es verdad.
- **Transposición:** Técnica de cifrado que consiste en alterar el orden de los símbolos en el mensaje a cifrar.
- **Sustitución:** Técnica de cifrado que consiste en alterar el orden de los símbolos del mensaje a cifrar.
- **Little Endian:** Formato de almacenamiento de estructuras de mayor longitud que el byte en un orden contrario al natural o de lectura.
- **Big Endian:** Formato de almacenamiento de estructuras de mayor longitud que el byte en orden natural o de lectura.
- **SGBD:** Software dedicado a servid de interfaz entre la base de datos, sus usuarios y las aplicaciones que hacen uso de ella.
- **BSD:** Berkeley Software Distribution (Distribución de Software de Berkeley). Licencia libre y permisiva, muy cercana al dominio público.
- **GNU GPL:** GNU General Public License (Licencia Pública de GNU). Licencia orientada a la libre distribución y uso del software.
- **NURBS:** Non-Uniform Rational B-Spline. Modelo matemático utilizado en el modelado gráfico para representar gráficamente curvas y superficies.
- **Autosemejanza:** Propiedad de los fractales en la que una pequeña sección puede ser vista como una réplica a menor escala de todo el fractal.
- **HLSL:** High Level Shader Languaje (Lenguaje de Shaders de Alto Nivel)**.**
- **GLSL:** OpenGL Shading Languahe (Lenguaje de Shaders de OpenGL).
- **FPS:** Frames Per Second (Frames Por Segundo). Tasa de frames por segundo utilizada con frecuencia para medir el rendimiento de una aplicación con representación gráfica tridimensional.
- **GPU:** Graphic Processing Unit (Graphics Processing Unit).
- **.NET:** Framework de Microsoft para desarrollo de aplicaciones de forma transparente a la plataforma.
- **C++:** Lenguaje de programación nacido como extensión de C para la manipulación de objetos.
- **Java:** Lenguaje de programación de alto nivel orientado a objetos.
- **Python:** Lenguaje de programación que soporta varios paradigmas (orientación a objetos, programación imperativa y funcional).
- **Perl:** Lenguaje de programación inspirado en C, en Lisp y AWK.
- **C#:** Lenguaje de programación orientado a objetos de la plataforma .NET.
- **GCC:** Colección de compiladores creada por el proyecto GNU de C, C++, Fortran, Ada, etc.
- **Xcode:** Entorno de desarrollo integrado de Apple.
- **Nvidia PhysX:** Middleware de Nvidia destinado a realizar cálculos matemáticos complejos en el ámbito de la simulación de físicas.
- **UDK:** Unreal Development Kit (Kit de Desarrollo de Unreal).
- **Windows Forms:** API de la plataforma .NET de Microsoft que provee de los elementos necesarios para la creación de interfaces.
- **SHA-512:** Secure Hash Algorithm (Algoritmo de Hash Seguro).
- **Captcha:** Completely Automated Public Turing Test to tell Computers And Humans Apart (Prueba Pública y Automatic para Diferenciar Máquinas y Humanos mediante Test de Turing).
- **System.Threading:** Librería de Microsoft que da soporte y ofrece los elementos necesarios para programar en multihilo.
- **MVC:** Model View Controller (Modelo Vista Controlador).

10. REFERENCIAS

[1] *La historia de Atari*. Recurso online consultado el 20 de Noviembre de 2010: http://www.neoteo.com/la-historia-de-atari

[2] *La historia de Nintendo*. Recurso online consultado el 22 de Noviembre de 2010: http://www.nintendo.es/NOE/es_ES/service/la_historia_de_nintendo_9911.html

[3] *Historia de Sega*. Recurso online consultado el 22 de Noviembre de 2010: http://sonicx2000.tripod.com/id2.html

[4] *Bulletin Board System*. Recurso online consultado el 30 de Noviembre de 2010: http://es.wikipedia.org/wiki/Bulletin_Board_System

[5] *Modos Simplex, Half-Duplex y Full-Duplex*. Recurso online consultado el 24 de Noviembre de 2010: http://www.eveliux.com/mx/modos-simplex-half-duplex-y-full-duplex.php

[6] *About ISO*. Recurso online consultado el 31 de Enero de 2011: http://www.iso.org/iso/home/about.htm

[7] *Protocolo de acuerdo a 3 vías*. Recurso online consultado el 2 de Febrero de 2011: http://neo.lcc.uma.es/evirtual/cdd/tutorial/transporte/twhandshake.html

[8] *Sobre el W3C*. Recurso online consultado el 22 de Febrero de 2011: http://www.w3c.es/Consorcio/

[9] *About the IETF*. Recurso online consultado el 22 de Febrero de 2011: http://www.ietf.org/about/

[10] *Conexiones persistentes (Keep-Alive).* Recurso online consultado el 23 de Febrero de 2011: http://trevinca.ei.uvigo.es/~txapi/espanol/proyecto/superior/memoria/node56.html

[11] *Sincronización entre procesos*. Recurso online consultado el 1 de Marzo de 2011: http://www.monografias.com/trabajos51/sincro-comunicacion/sincro-comunicacion.shtml

[12] *Hibernate*. Recurso online consultado el 19 de Mayo de 2011: http://es.wikipedia.org/wiki/Hibernate

[13] *Logaritmo discreto*. Recurso online consultado el 23 de Marzo de 2011: http://matematica.laguia2000.com/general/logaritmo-discreto

[14] *Ataque Man-in-the-middle*. Recurso online consultado el 23 de Marzo de 2011: http://es.wikipedia.org/wiki/Ataque_Man-in-the-middle

[15] Fundamentos del mapeado UV. Recurso online consultado el 20 de Noviembre de 2010: http://www.yetze.ro/tutoriales-3d/fundamentos-del-mapeado-uv/

[16] *Perlin Noise: Funciones pseudoaleatorias para generación de terrenos*. Recurso online consultado el 21 de Noviembre de 2011: http://piziadas.com/es/2010/04/perlin-noise-funciones-pseudoaleatorias.html

[17] *Introducción a Direct3D*. Recurso online consultado el 20 de Noviembre de 2010: http://gsii.usal.es/~igrafica/descargas/temas/Tema14.pdf

[18] *OpenGL – The Industry's Foundation for High Perfomance Graphics*. Recurso online consultado el 28 de Agosto de 2012: http://www.opengl.org/about/

[19] Skybox (2D). Recurso online consultado el 28 de Agosto de 2012: https://developer.valvesoftware.com/wiki/Skybox_(2D)

[20] *SHA – Secure Hash Algorithm*. Recurso online consultado el 25 de Agosto de 2012: https://www.ccn-cert.cni.es/publico/serieCCN-STIC401/es/s/sha.htm

[21] *Demonios*. Recurso online consultado el 25 de Agosto de 2012: http://es.wikipedia.org/wiki/Demonio_(inform%C3%A1tica)

[22] **Esmitt Ramírez**. *Frustrum Culling*. Recurso online consultado el 26 de Agosto de 2012: http://unsitioweb.com/man_juegos/frustumculling2.pdf

[23] **Juan Pavón Mestras**. *Estructura de las Aplicaciones Orientadas a Objetos; El patrón Modelo-Vista-Controlador*. Recurso online consultado el 22 de Abril de 2011: http://www.fdi.ucm.es/profesor/jpavon/poo/2.14.MVC.pdf

[24] *Patron State*. Recurso online consultado el 7 de Julio de 2011: http://patronesdediseno.blogspot.com.es/2009/05/patron-state.html

[25] 3D Models with .x files. Recurso online consultado el 22 de Agosto de 2012: http://www.toymaker.info/Games/html/3d_models.html

[26] *Pool de conexiones*. Recurso online consultado el 10 de Agosto de 2011: http://www.gxtechnical.com/gxdlsp/pub/genexus/java/docum/manuals/8.0/mjavaf5.htm

[27] *Bloquear Lecturas SELECT…FOR UPDATE*. Recurso online consultado el 25 de Mayo de 2011: dev.mysql.com/doc/refman/5.0/es/innodb-locking-reads.html

[28] *Skeletal animation*. Recurso online consultado el 29 de Agosto de 2012: http://en.wikipedia.org/wiki/Skeletal_animation

[29] **Dean Sekulic**. *Efficient Occlusion Culling*. GPU Gems. Recurso online consultado el 29 de Agosto de 2012: http://http.developer.nvidia.com/GPUGems/gpugems_ch29.html

[30] **Symeon Xenitellis**. The Open-source PKI Book. Recurso online consultado el 28 de Agosto de 2012: http://ospkibook.sourceforge.net/docs/OSPKI-2.4.7/OSPKI-html/ospki-book.htm

[31] *Man In The Middle*. Recurso online consultado el 28 de Agosto de 2012: http://www.flu-project.com/man-in-the-middle.html